CHAPTER 8 Logic and circuit simulation

Jiun-Lang Huang
National Taiwan University, Taipei, Taiwan

Cheng-Kok Koh
Purdue University, West Lafayette, Indiana

Stephen F. Cauley
Purdue University, West Lafayette, Indiana

ABOUT THIS CHAPTER

Logic simulation and circuit simulation are typically used in conjunction with functional verification to verify the correctness of an integrated circuit. During the logic design stage, designers rely on logic simulation to verify whether the design meets its specifications and contains any design errors. During the circuit design stage, designers use circuit simulation to test and characterize digital cell libraries, memory models, and ***analog and mixed-signal*** (AMS) circuits that require detailed timing analysis to ensure the correct operation of these circuits.

This chapter begins with a discussion of logic simulation. After an introduction to the logic circuit models, the popular compiled-code and event-driven logic simulation techniques are described. This is followed by hardware-accelerated logic simulation that is commonly referred to as hardware emulation and is intended to bridge the growing gap between circuit complexity and software simulator efficiency. Commonly used hardware emulation techniques are introduced first, followed by a description of the two crucial ingredients of emulators: reconfigurable computing units and interconnection architectures. The second half of the chapter is devoted to circuit-level simulation. After describing the circuit simulation models and essential numerical methods, the chapter explains the procedures required to simulate ***very large-scale integration*** (VLSI) circuits with interconnects and nonlinear devices. By working through this chapter, the reader will learn about the major logic simulation, hardware emulation, and circuit simulation techniques. This background will be valuable in selecting the simulation method that best meets the design needs.

8.1 INTRODUCTION

Simulation empowers a designer to predict a design's behavior without physically implementing it. In the design phase, the main purpose of simulation is **design verification**. Figure 8.1 depicts the flow of using simulation for design verification. During each design stage, the functional specification documents the required functionality and performance for the design and a corresponding circuit description is generated in conformance with the given specification. To ensure conformance, verification testbenches consisting of a set of input stimuli and expected output responses are created. The simulator then takes the circuit description and the input stimuli as inputs and produces the simulated responses. Any discrepancy between the simulated and expected responses (detected by the response analysis process) indicates that redesign or modification is necessary. Once the circuit has been verified to an acceptable confidence level, the design process advances to the next design stage.

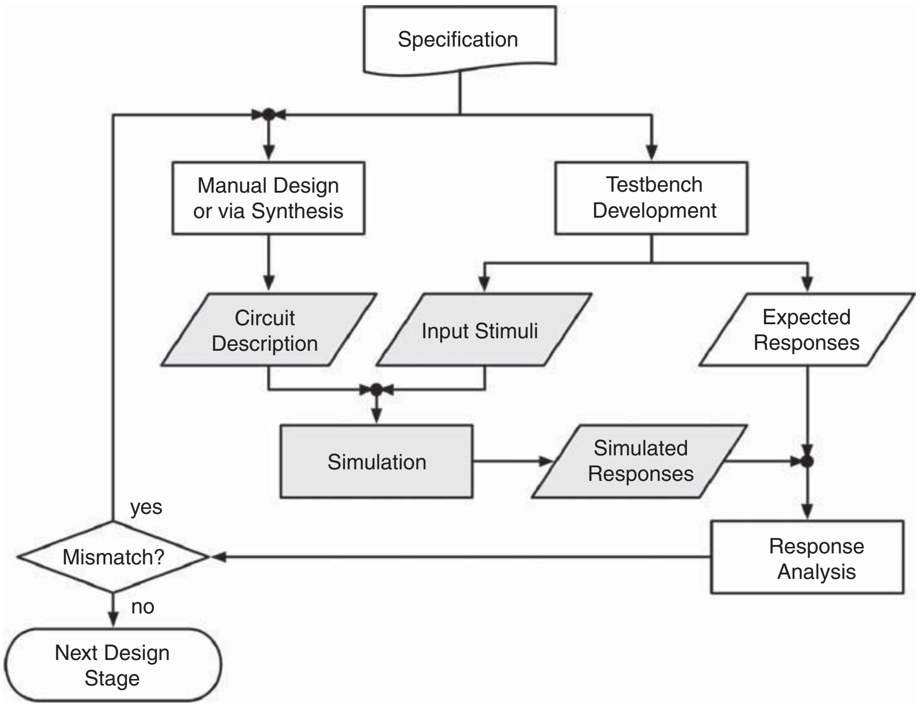


FIGURE 8.1

Simulation for design verification.

Table 8.1 Design Descriptions at Different Circuit Abstraction Levels

Design Stage	**Circuit Description**
Behavioral Level or Electronic System Level (ESL)	C/C++, SystemC [SystemC 2008] SystemVerilog [SystemVerilog 2008]
Register-Transfer Level (RTL)	Verilog [IEEE 1463–2001] [Thomas 2002] VHDL [IEEE 1076–2002]
Gate Level	Gate-Level Netlists
Circuit Level	Transistor-Level Schematics

8.1.1 **Logic simulation**

Digital circuit simulation can be performed at different abstraction levels—from the highest behavioral level to the lowest device level. At each level, a suitable description language that captures the required functional specification is used to describe the design. Table 8.1 lists the commonly used abstraction levels and the corresponding circuit descriptions.

In general, design verification begins at the behavioral level or *electronic-system level* (ESL) where the algorithm correctness and system throughput are the major concerns. Then, at the ***register-transfer level*** (RTL), the design is described in terms of blocks such as registers, counters, data processing units, and controllers, as well as the data/control flow between these blocks. Because ESL/RTL verification usually does not involve detailed timing analysis, ESL or RTL design verification is also referred to as **functional verification** [Wile 2005].

Logic/scan synthesis comes into play after the RTL design stage. The gate-level netlist corresponding to the RTL design, which includes scan cells, is synthesized with logic elements provided in a cell library. Finally, the transistor-level description provides the most accurate model of the design. However, because it is much slower than gate-level simulation, transistor-level simulation is usually used only for characterizing timing critical paths and library cells.

This chapter first discusses gate-level logic simulation. Although digital circuits use two-valued Boolean algebra as the underlying mathematics, most logic simulators include two more values, unknown (u) and high-impedance (Z), to handle the inevitable uncertainties in practical circuits. Understanding the capabilities and limitations of the 4-valued logic system prevents the users from incorrectly interpreting the simulation results. Furthermore, to deal with timing, delay information must be incorporated into the logic element descriptions. Thus, both gate and wire delays must be taken into account for modern designs. The two major logic simulation techniques are **compiled-code simulation** and **event-driven simulation**. Grounded in distinct principles, each of them has its own advantages and drawbacks and finds applications in different areas of the design process.

8.1.2 Hardware-accelerated logic simulation

As the circuit complexity continues growing, logic simulation becomes the bottleneck of design verification—available logic simulators are too slow for practical *system-on-chip* (SOC) designs or *hardware/software* (HW/SW) **co-simulation** applications. Several types of **hardware-accelerated logic simulation** techniques have been proposed, including **simulation acceleration**, **(in-circuit) emulation**, and **hardware prototyping**, each of which has its advantages and shortcomings. A modern emulator may be a hybrid of the preceding types or be able to execute several types to meet the requirements of different design stages.

Most emulator systems consist of arrays of reconfigurable logic computing units that are directly or indirectly interconnected. Although ***field programmable gate array*** (FPGA) is a natural choice for the computing unit, the emulation system performance is severely limited by the available *input/output* (I/O) pins. Indirect interconnect architectures such as full and partial crossbars and time-multiplexed I/O are possible solutions to improve the inter-chip data bandwidth. Other approaches include exploring different use models of FPGA and the use of programmable processors as the reconfigurable computing units.

8.1.3 Circuit simulation

Circuit simulation is an increasingly indispensable tool for the design of ***integrated circuits*** (ICs). The turnaround time and cost of fabrication, along with the sheer number of design parameters under consideration, prevent circuit designers from relying on intuition and extensive experimentation to meet their design specifications. Instead, designers can use an understanding of the dynamic behavior for their circuits, learned through circuit simulation, to save both time and resources for fabrication.

The simulation of ICs is a very structured area that is grounded in the first principles of **current** and **voltage** relationships. The process of simulating a circuit begins with the "modeling" of each element from the circuit in terms of basic building blocks such as current and voltage sources, resistors, capacitors, and inductors. The parameters for each element in the model may be time-varying or time-invariant. The goal of these models is to accurately mimic the dynamic behavior of the elements while providing the simplest possible representation. Specifically, with these models, the designer can easily construct a set of current and voltage relationships that describe the behavior of the entire circuit.

There are several different representations of circuit equations that primarily rely on the ***Kirchhoff's voltage law*** (KVL) and ***Kirchhoff's current law*** (KCL) in the formulations. Although the behavior of some very simple circuits can be described analytically, we must investigate general numerical techniques for even modest sized ICs. The scalability of these techniques is crucial when considering the growth of modern designs. Thus, in practice, the modeling of circuit elements and the subsequent formulation of circuit equations should be performed while keeping in mind the computation load required for the resulting numerical techniques.

8.2 LOGIC SIMULATION MODELS

In this section, we discuss the gate-level simulation models for combinational and sequential networks, which have widespread acceptance in the integrated circuit community.

8.2.1 Logic symbols and operations

In addition to 1 and 0, logic simulators often include two more symbols: u (unknown) and Z (high-impedance); the former represents the uncertain circuit behavior, and the latter helps resolve the behavior of tristate logic. For cases in which 0, 1, u, and Z are insufficient to meet the required simulation accuracy, intermediate logic states that incorporate both value and strength may be used.

8.2.1.1 *"1" and "0"*

The basic mathematics for most digital systems is the two-valued Boolean algebra. In two-valued Boolean algebra, a variable can assume only one of the two values, *true* or *false*, which are represented by the two symbols 1 and 0, respectively. Note that 1 and 0 here do not represent numerical quantities. Physical representations of the two symbols depend on the logic family of choice. Consider the most popular CMOS logic as an example; the two symbols 1 and 0 represent two distinct voltage levels, *power* (V_{DD}) and *ground* (V_{SS}), respectively. Here we assume positive logic is used. Whether a signal's value is 1 or 0 depends on which voltage source it is connected to.

8.2.1.2 *The unknown value* u

Almost all practical digital circuits contain memory elements (*e.g.*, flip-flops and memories) to store the circuit state; however, when these circuits are powered up, the initial states of their memory elements are usually unknown. To handle such situations, the logic symbol u is introduced to indicate an *unknown* logic value. By associating u with a signal, we mean that the signal is 1 or 0, but we are not sure which one is the actual value.

8.2.1.3 *The high-impedance state* Z

Until now, the logic signal states that we have discussed are 1 and 0, indicating that the signal is connected to either V_{DD} or V_{SS}. (The unknown symbol indicates uncertainty; however, the signal of interest is still 1 or 0.) In addition to 1 or 0, tristate gates have a third, high-impedance state, denoted by logic symbol Z. Tristate gates permit several gates to time-share a common wire, called a *bus*. A signal is in the Z state if it is connected to neither V_{DD} nor V_{SS}.

Figure 8.2 depicts a typical bus application. In this example, three bus drivers (G_1, G_2, and G_3) drive the bus wire y. Each driver G_i is controlled by an *enable* signal e_i, and its output o_i is determined as follows:

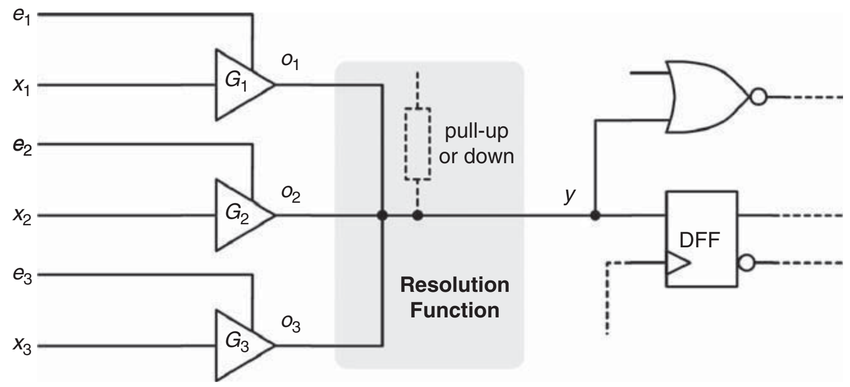


FIGURE 8.2

A tristate circuit example.

$$o_i = \begin{cases} x_i & \text{if } e_i = 1 \\ Z & \text{if } e_i = 0 \end{cases}$$

When $o_i = Z$, G_i has no effect on the bus wire y, leaving the control to other drivers.

A **bus conflict** occurs if at least two drivers drive the bus wire to opposite binary values. Such situations may cause the circuit to be permanently damaged. In addition to design errors, abnormal bus states could occur during testing when the circuit is not in its normal operating environment and may receive illegal input sequences. On the other hand, if no driver is activated, the bus is in a **floating state,** because it is not connected to V_{DD} or V_{SS}. A pull-up or pull-down network that connects the bus to V_{DD} or V_{SS} by means of a resistor may be added to provide a default 1 or 0 logic value (Figure 8.2); otherwise, the bus wire will retain its previous value as a result of trapped charge in the parasitic wire capacitance. Because the charge could decay in time, a **bus keeper** consisting of two weak inverters is usually added to the bus wire to keep its previous value as long as the bus is in the floating state.

8.2.1.4 ***Basic logic operations***

The input/output relationships of the three basic logic operations (AND, OR, and NOT) that use the four logic symbols (0, 1, u, and Z) are summarized in Table 8.2. It is worth noting that:

1. Simulation results based on these truth tables are pessimistic (*i.e.*, a signal may be reported as unknown even though its value can be uniquely determined as 0 or 1 [Breuer 1972]).
2. The four symbols are insufficient to handle intermediate signal values (*e.g.*, values in the middle of V_{DD} and V_{SS}, which may occur in tristate buses, switch-level networks, or defective circuits). To enhance the

Table 8.2 Truth Tables of AND, OR, and NOT

AND	**0**	**1**	***u***	***z***	**OR**	**0**	**1**	***u***	***z***	**NOT**	**0**	**1**	***u***	***z***
0	0	0	0	0	**0**	0	1	*u*	*u*		1	0	*u*	*u*
1	0	1	*u*	*u*	**1**	1	1	1	1					
u	0	*u*	*u*	*u*	***u***	*u*	1	*u*	*u*					
z	0	*u*	*u*	*u*	***z***	*u*	1	*u*	*u*					

```
table
      //  select   a     b     :    out
      //  0        0     ?     :    0
      //  0        1     ?     :    1
      //  1        ?     0     :    0
      //  1        ?     1     :    1
      //  ?        0     0     :    0
      //  ?        1     1     :    1
endtable
```

FIGURE 8.3

The truth table of a two-input multiplexer.

simulation resolution, intermediate logic states that incorporate both signal value and strength may be used [Miczo 2003].

To support customized logic elements with complex sequential or combinational behavior, modern simulation tools support ***user-defined primitives*** (UDPs). Figure 8.3 shows how Verilog models a two-input multiplexer with a truth table. Here, *a* and *b* are the data inputs, *select* is the control input, and *out* is the multiplexer output. The symbol "?" is a shorthand notation for 0, 1, or *u*.

8.2.2 Timing models

Delay is a fact of life for all electrical components, including logic gates and interconnection wires. In this section, we discuss the commonly used gate and wire delay models.

8.2.2.1 *Transport delay*

The **transport delay** refers to the time duration it takes for the effect of gate input changes to appear at gate outputs. Several transport delay models characterize this phenomenon from different aspects.

The **nominal delay** model specifies the same delay value for the output rising and falling transitions. Consider the AND gate *G* in Figure 8.4 as an example. Here *B* is fixed at 1; thus, the output of *G* is only affected by *A*. Assuming that *G* has a nominal delay of $d_N = 2$ ns and *A* is pulsed to 1 for 1 ns, the

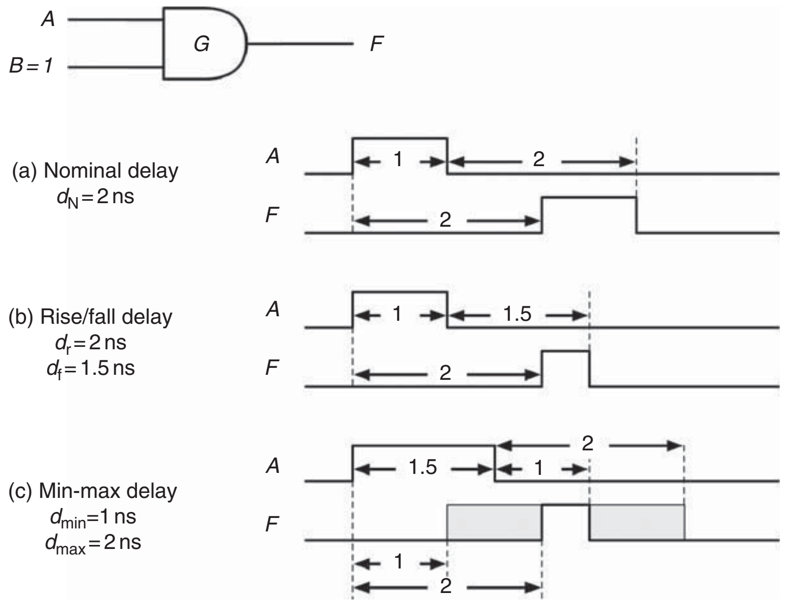


FIGURE 8.4

Transport delay models.

corresponding simulation result is shown in Figure 8.4a. Under the nominal delay model, the output waveform at *F* is simply a version of *A* delayed by 2 ns.

For cases in which the rising and falling times are different (*e.g.*, the pull-up and pull-down transistors of the gate have different driving strengths), one may opt for the **rise/fall delay** model. In Figure 8.4b, the setup is the same as that in Figure 8.4a except that the rise/fall delay model is used instead; the rise and fall delays are $d_r = 2$ ns and $d_f = 1.5$ ns, respectively. Because of the difference between the two delays, the duration of the output pulse shrinks from 1 to 0.5 ns.

If the gate transport delay cannot be uniquely determined (*e.g.*, because of process variations), one may use the **min–max delay** model. In the min–max delay model, the minimum and maximum gate delays (d_{min} and d_{max}) are specified to represent the ambiguous time interval in which the output change may occur. In Figure 8.4c, the minimum and maximum delays are 1 and 2 ns, respectively, and a 1.5-ns pulse is applied at *A*. In response to the delay uncertainty, two ambiguous intervals (the shaded regions), corresponding to the rising and falling transitions, are observed at output *F*. Within the two ambiguous intervals, the exact output value is unknown.

8.2.2.2 *Inertial delay*

The **inertial delay** is defined as the minimum input pulse duration necessary for the output to switch states. Pulses shorter than the inertial delay cannot pass

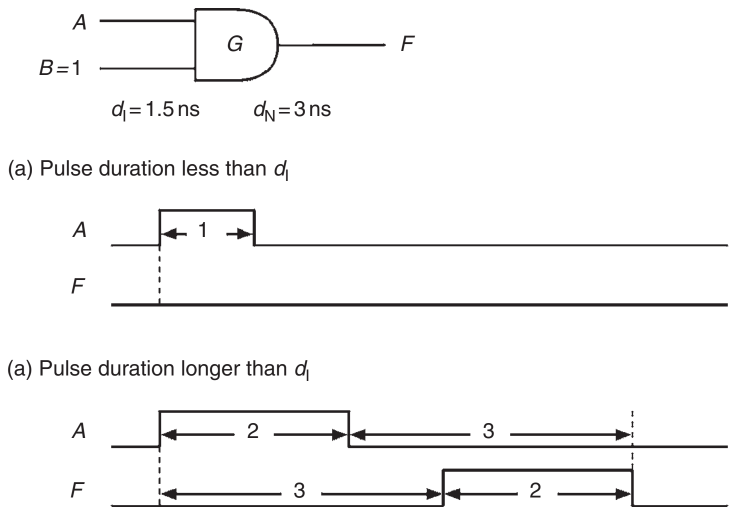

A
B=1
G
F
d_I=1.5 ns
d_N=3 ns
(a) Pulse duration less than d_I
A
1
F
(a) Pulse duration longer than d_I
A
2
3
F
3
2

FIGURE 8.5

Inertial delay.

through the circuit element. The inertial delay models the limited bandwidth of logic gates. Figure 8.5 illustrates this filtering effect. Assume that the AND gate has an inertial delay (d_I) of 1.5 ns and a nominal delay of 3 ns. Let us fix B at 1 and apply a pulse on A. In Figure 8.5a, the 1-ns pulse is filtered and the output remains at a constant 0. In Figure 8.5b, the pulse is long enough (2 ns) and an output pulse is observed 3 ns later.

8.2.2.3 *Functional element delay model*

Functional elements, such as flip-flops, have more complicated behaviors than simple logic gates and require more sophisticated timing models. In Table 8.3, the I/O delay model of the positive-edge-triggered D flip-flop is depicted. Take the asynchronous preset operation (second row) as an example. Regardless of the *Clock* and *D* values, if the current flip-flop state (q) is 0 and *ClearB* remains 1, changing *PresetB* from 1 to 0 (denoted by the down arrow) will cause output transitions at *Q* and *QB* after 1.6 and 1.8 ns, respectively. Besides the input-to-output transport delay, the flip-flop timing model usually contains timing constraints, such as setup/hold times and inertial delays for each input.

8.2.2.4 *Wire delay*

Figure 8.6a illustrates the distributed RLC model of a metal wire. In the presence of the passive components, it takes finite time, called the **propagation delay**, for a signal to travel from point p to point q.

In general, wire delays are specified for each connected gate output and gate input pair, because the physical distances and thus the propagation delays between the driver and receiver gates vary. In Figure 8.6b, the inverter output

Table 8.3 The D Flip-Flop I/O Delay Model

Input Condition				Present State	Outputs		Delay (ns)		
D	*Clock*	*PresetB*	*ClearB*	*q*	*Q*	*QB*	to *Q*	to *QB*	Comments
X	*X*	↓	1	0	↑	↓	1.6	1.8	Asynchronous preset
X	*X*	1	↓	1	↓	↑	1.8	1.6	Asynchronous clear
1	↑	1	1	0	↑	↓	2	3	*Q*: 0→1
0	↑	1	1	1	↓	↑	3	2	*Q*: 1→0

Note: X indicates "don't care."

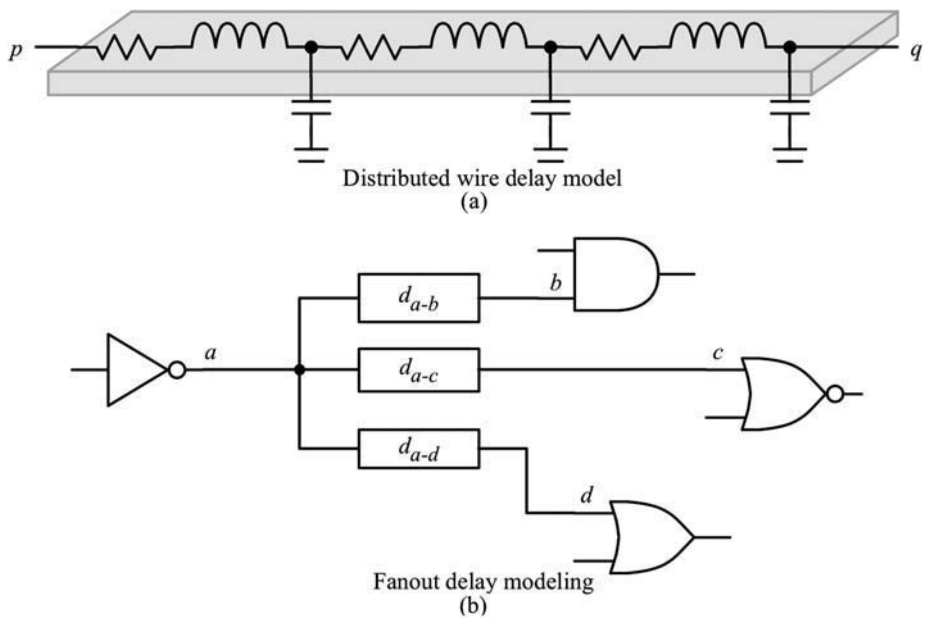


FIGURE 8.6

Wire delay model.

a branches out to drive three gates. To model the wire delays associated with the three signal paths, one may insert delay elements d_{a-b}, d_{a-c}, and d_{a-d} into the fanout branches. For convenience, wire delays may also be viewed as the receiver gate input delays and become part of the receiver gate delay model.

8.3 LOGIC SIMULATION TECHNIQUES

The general model of a gate-level or RTL network consists of the combinational network and memory elements as depicted in Figure 8.7. In this figure, X and Z denote the *primary inputs* (PI) and *primary outputs* (PO), and Q and Q^+ denote the present and next states of the flip-flops. Q and Q^+ are also called the *pseudo primary inputs* (PPI) and *pseudo primary outputs* (PPO), respectively.

For ease of illustration, we assume that a single clock and D-type flip-flops are used. The simplified synchronous sequential circuit simulation flow is depicted in Figure 8.8. At the beginning of each clock cycle, the simulator evaluates the flip-flops and then makes the necessary updates (*i.e.*, $Q \leftarrow Q^+$). Then, the input vector is read in and the combinational part is evaluated. The simulation continues until the specified simulation time has been reached or there is no more input vector left.

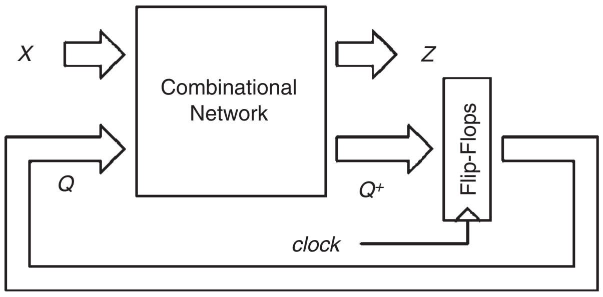


FIGURE 8.7

A general sequential circuit model.

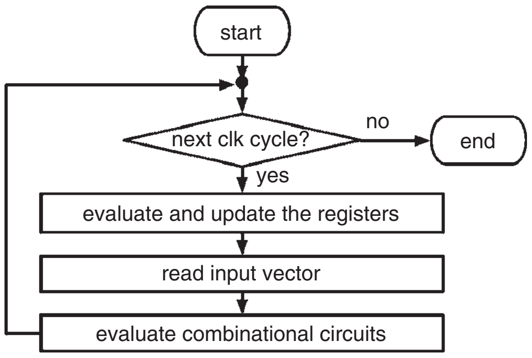

start
next clk cycle?
no
end
yes
evaluate and update the registers
read input vector
evaluate combinational circuits

FIGURE 8.8

The simplified synchronous circuit simulation flow.

In this section, we will discuss two commonly used logic simulation methods: *compiled-code simulation* and *event-driven simulation*. Although the circuits are combinational in the discussions, these techniques can be easily extended to deal with sequential circuits.

8.3.1 Compiled-code simulation

The idea of *compiled-code simulation* is to translate the digital circuit into a series of machine instructions that model the functions of individual gates and the interconnects between them.

8.3.1.1 *Preprocessing*

In practice, logic optimization and levelization are performed before the actual code generation process. The purpose of **logic optimization** is to enhance the simulation efficiency. A typical optimization process consists of the transformations illustrated in Figure 8.9 [Wang 1987]. Because each gate corresponds to one or more statements in the compiled code, logic optimization reduces the program size and execution time.

To avoid unnecessary computations, logic gates must be evaluated in an order such that a gate will not be evaluated until all its driving gates have been evaluated. The logic levelization algorithm starts by assigning all the PI's and

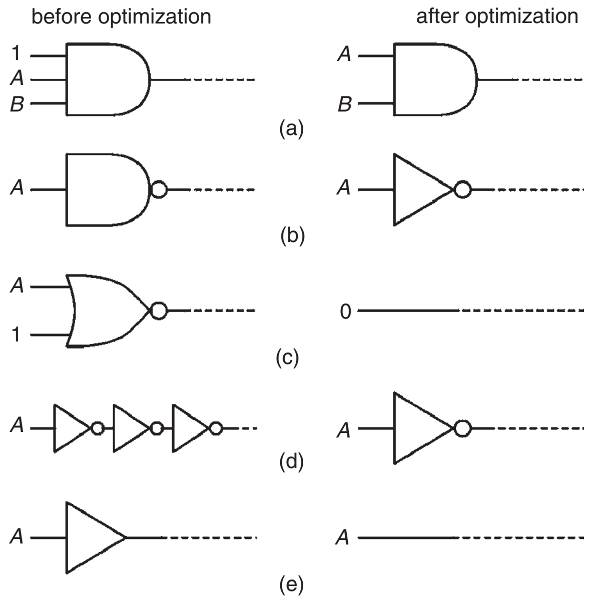


FIGURE 8.9

Logic optimization for compiled-code simulation.

PPI's level 0. For each logic element, its level is equal to the maximum of its driving elements' levels plus 1.

8.3.1.2 *Code generation*

Depending on performance, portability, and maintainability needs, different code generation techniques may be used [Wang 1987]. The three approaches are interpreted code, high-level programming language source code, and native machine code.

In the interpreted code approach, the target machine is a software emulator. During simulation, the instructions are interpreted and executed one at a time. This approach offers the best portability and maintainability at the cost of reduced performance. Mapping the simulated digital network to a high-level programming language such as C enhances performance and is portable to any target machine that has a C compiler. The compilation time could be a severe limitation for fault simulators that require recompilation for each faulty circuit. The native machine code approach generates the native machine code directly without the need of compilation; this makes it a more viable solution to fault simulation. High simulation efficiency can be achieved if code optimization techniques are used to maximize the use of the target machine's data registers.

Take the network in Figure 8.10 as an example. The generated pseudocode is shown in the following. In the actual implementation, each statement is replaced with the corresponding language constructs or machine instructions.

```
while true
  read(A, B, C);
  E = OR(B, C);
  H = AND(A, E);
  J = NOT(E);
  K = NOR(H, J);
  output(K);
end
```

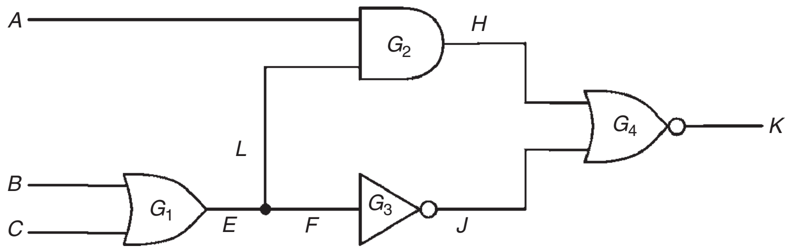


FIGURE 8.10

An example network for native machine code generation.

8.3.1.3 *Applications*

The main limitation of compiled-code simulation is that it is incapable of timing modeling. As a result, it fails to detect timing problems such as glitches and race conditions. Despite the limitations, compiled-code simulation finds its application in **cycle-based simulation,** where only the logic function correctness is of interest and zero-delay model is used. Compiled-code simulation is most effective when binary logic simulation suffices. In such cases, machine instructions are readily available for Boolean operations (*e.g*., AND, OR, and NOT). Further speedup is possible with the bit-wise logic operation instructions that allow concurrent simulation of independent vectors or vector sequences. With 3 or more logic symbols, which is usually the case, the logic evaluation processes are more complicated but still manageable.

8.3.2 Event-driven simulation

In contrast to compiled-code simulation, *event-driven simulation* exhibits high simulation efficiency by performing gate evaluations only when necessary. We will use Figure 8.11 to illustrate the event-driven simulation concept. In this example, two consecutive input patterns $ABC = 001$ and 111 are applied to the circuit, and the corresponding circuit values are shown. Note that the application of the second vector does not change the input of G_3, so G_3 is not evaluated for the second vector. In event-driven simulation, the switching of a signal's value is called an **event**, and an event-driven simulator monitors the occurrences of events to determine which logic elements to evaluate.

8.3.2.1 *Zero-delay event-driven simulation*

Figure 8.12 depicts the zero-delay event-driven simulation flow. (A zero-delay simulation is one in which gates and interconnections are assumed to have zero delay.) At the beginning of the simulation flow, the initial signal values, which may be given or simply unknown, are read in and assigned. Then, a new input vector is loaded and the primary inputs at which events occur (called active PIs) are identified. To propagate the events toward primary outputs, gates driven by

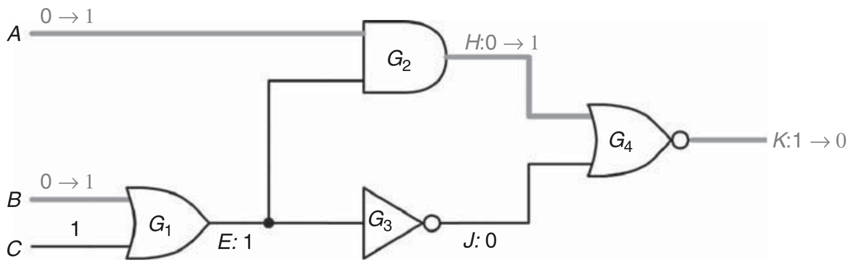


FIGURE 8.11

Signal transitions between consecutive input vectors.

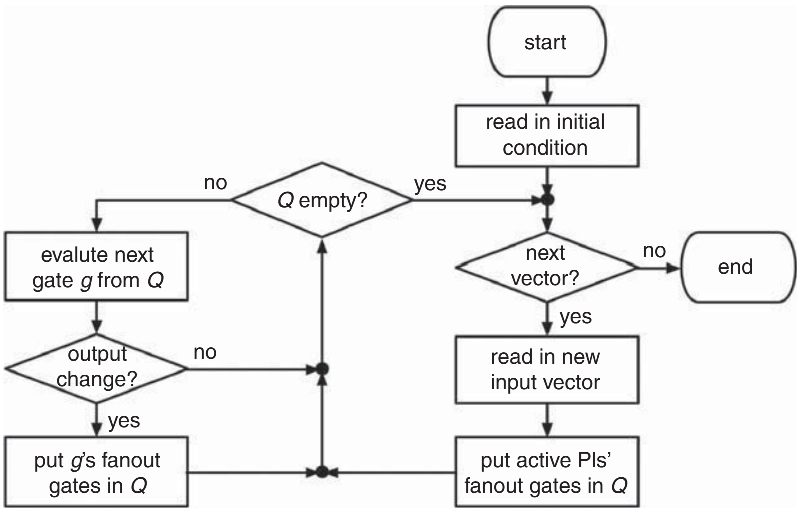

start
read in initial condition
Q empty?
no
yes
evalute next gate g from Q
next vector?
no
end
yes
output change?
no
yes
read in new input vector
put g's fanout gates in Q
put active PIs' fanout gates in Q

FIGURE 8.12

The zero-delay event-driven simulation flow.

active primary inputs are put in the event queue Q, which stores the gates to be evaluated. As long as Q is not empty, a gate g is dequeued from Q and evaluated. If the output of g changes (*i.e.*, a new event occurs), the fanout gates of g are placed in Q. When Q becomes empty, the simulation for the current input vector is finished, and the simulator proceeds to process the next input vector.

8.3.2.2 *Nominal-delay event-driven simulation*

The greatest advantage of event-driven simulation over compiled-code simulation is that it can handle any delay model. A sophisticated **event scheduler** keeps track of event occurrences and schedules the necessary gate evaluations at the proper time points. Because events must be evaluated in chronological order, the scheduler is implemented as a priority queue.

Figure 8.13 depicts one possible priority queue implementation for a nominal delay event-driven simulator. In the priority queue, the vertical list is an ordered list that stores the time stamps when events occur. Attached to each time stamp t_i is a horizontal list of events that occur at time t_i. During simulation, a new event that will occur at time t_i is appended to the event list of time stamp t_i. For example, in Figure 8.13, the value of signal w will switch to v_w^+ at t_i. If t_i is not in the time stamp list yet, the scheduler will first place it in the list according to the chronological order.

For the priority queue scheduler in Figure 8.13, the time needed to locate a time stamp to insert an event grows with the circuit size. To improve the event scheduler efficiency, one may use, instead of a linked list, an array of evenly spaced time stamps. Although some entries in the array may have empty event

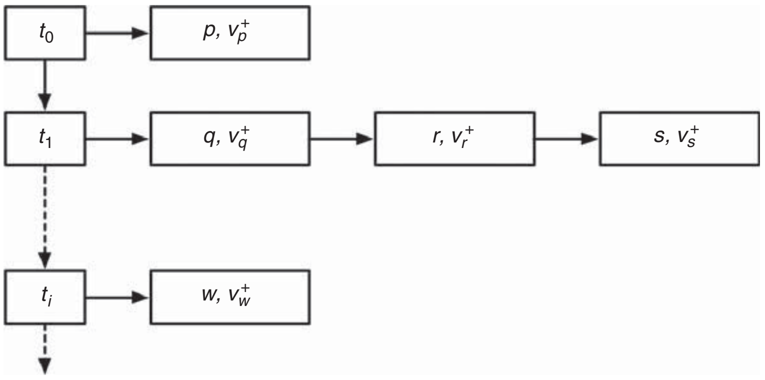


FIGURE 8.13

Priority queue event scheduler.

lists, the overall search time is reduced, because the target time stamp can be indexed by its value. Further enhancement is possible with the concept of **timing wheel** [Ulrich 1969]. Let the time resolution be one time unit and the array size M. A time stamp that is d time units ahead of current simulation time (with array index i) is stored in the array and indexed by $(i + d)$ modulo M if d is less than M; otherwise, it is stored in an overflow remote event list similar to that shown in Figure 8.13. The array is referred to as the timing wheel because of the modulo-M-induced circular structure. Remote event lists are brought into the timing wheel once their time stamps are within M-1 time units from current simulation time.

A two-pass strategy for nominal delay event-driven simulation is depicted in Figure 8.14. When there are still events with future time stamps to process, the event list L_E of next time stamp t is retrieved. L_E is processed in a two-pass manner. In pass one (the left shaded box), the simulator determines the set of gates to be evaluated. The notation (g, v_g^+) indicates that the output of gate g is to become v_g^+. For each event (g, v_g^+), if v_g^+ is the same as g's current value v_g, this event is false and is discarded. On the other hand, if $v_g^+ \neq v_g$, *i.e.*, (g, v_g^+) is a valid event, then v_g is updated to v_g^+, and the fanout gates of g are appended to the activity list L_A. In the second pass (the right shaded box), gates are evaluated and new events are scheduled. As long as the activity list L_A is non-empty, a gate g is retrieved and evaluated. Let the evaluation result be v_g^+. The scheduler will schedule the new event (g, v_g^+) at time stamp t + delay(g), where delay(g) denotes the nominal delay of gate g. The two-pass strategy avoids repeated evaluation of gates with events on multiple inputs.

In the following, we will use the circuit in Figure 8.10 to demonstrate the two-pass event-driven strategy. In this example, the nominal delays for G_1, G_2, G_3, and G_4 are 8, 8, 4, and 6 ns, respectively, and there are four input events: $(A, 1, 0)$, $(C, 0, 2)$, $(B, 0, 4)$, and $(A, 0, 8)$, where the notation (w, v_w^+, t)

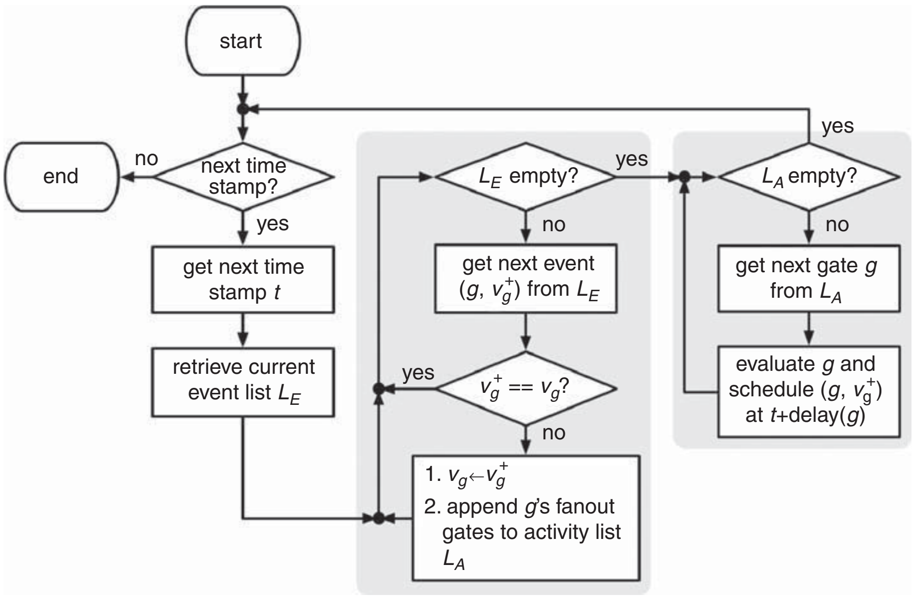

start
end
no
next time stamp?
yes
get next time stamp t
retrieve current event list L_E
yes
L_E empty?
yes
no
get next event (g, v_g^+) from L_E
yes
$v_g^+ == v_g$?
no
1. $v_g \leftarrow v_g^+$
2. append g's fanout gates to activity list L_A
yes
L_A empty?
no
get next gate g from L_A
evaluate g and schedule (g, v_g^+) at t+delay(g)

FIGURE 8.14

The two-pass nominal-delay event-driven simulation flow.

represents the event that signal w switches to v_w^+ at time t. The simulation progress is shown in Table 8.4. At time 0, there is only one primary input event (A, 1). Because A drives G_2, G_2 is added to activity list L_A. Evaluation of G_2 returns $H = 1$; therefore, the event (H, 1) is scheduled at time 8 (*i.e.*, 8 ns, the delay of G_2 after the current time). At time stamps 2 and 4, the two input events at C and B are processed in the same way. There are two events at time 8: the input event (A, 0) and the scheduled event (H, 1) from time stamp 0. Because both events are valid, the two affected gates, G_2 and G_4, are put in L_A for evaluation. The corresponding events (H, 0) and (K, 0) are scheduled at times 16 and 14, respectively. Note that the event (E, 1) at time 10 is false, because it does not cause a signal transition; therefore, no gate evaluation is performed.

8.4 HARDWARE-ACCELERATED LOGIC SIMULATION

As the IC density and complexity continue growing, verifying the correctness of a new design before its first silicon has become the key to success. Although versatile and accurate, logic simulation is too slow for large designs, not to mention SOC designs that necessitate *hardware/software* (HW/SW) co-simulation.

Various hardware-acceleration techniques have been developed to bridge the gap between the IC complexity and logic simulation efficiency. A simplified block diagram of an FPGA-based hardware emulator is illustrated in Figure 8.15.

Table 8.4 A Two-Pass Event-Driven Simulation Example

Time	L_E	L_A	Scheduled Events
0	$\{(A,1)\}$	$\{G_2\}$	$\{(H,1,8)\}$
2	$\{(C,0)\}$	$\{G_1\}$	$\{(E,1,10)\}$
4	$\{(B,0)\}$	$\{G_1\}$	$\{(E,0,12)\}$
8	$\{(A,0),(H,1)\}$	$\{G_2,G_4\}$	$\{(H,0,16),(K,0,14)\}$
10	$\{(E,1)\}$		
12	$\{(E,0)\}$	$\{G_2,G_3\}$	$\{(H,0,20),(J,1,16)\}$
14	$\{(K,0)\}$		
16	$\{(H,0),(J,1)\}$	$\{G_4\}$	$\{(K,0,22)\}$
20	$\{(H,0)\}$		
22	$\{(K,0)\}$		

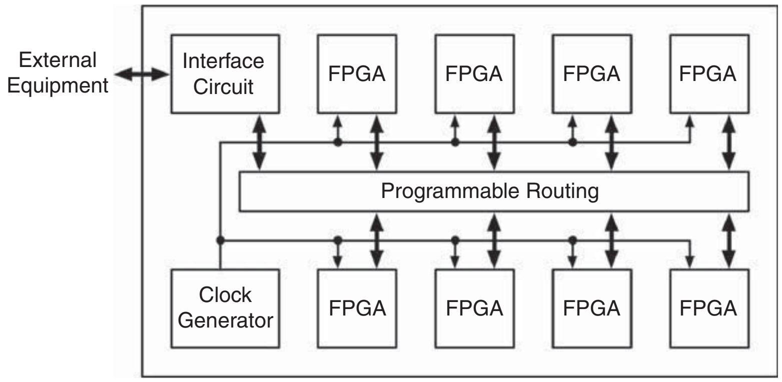


FIGURE 8.15

A simplified hardware emulator block diagram.

(In some hardware emulators, programmable ***application-specific integrated circuits*** [ASICs] are used instead of FPGAs.) The hardware emulator (called emulator hereafter) includes a circuit board holding a set of FPGAs, a routing system, and an interface circuit. For the emulator to emulate an IC, an external host computer programs each FPGA to emulate a portion of the IC and programs the routing system to route inter-FPGA signals. External test equipment or the target system board can then verify the emulated IC by supplying test signals to the FPGAs and monitoring the output responses from the FPGAs by means of the interface circuit.

The fundamental differences between software simulators and hardware emulators are as follows.

1. A logic simulator executes the RTL code or evaluates the logic elements serially. An emulator, on the other hand, executes the whole design concurrently or in massive parallelism.
2. Logic simulators are more flexible in terms of supported logic symbols and timing models. Emulators are basically 2-state machines and are more suitable for verifying logic correctness (*e.g.*, cycle-based simulation).
3. Logic simulators support a rich set of debugging capabilities. For example, one can stop the design at the middle of a cycle or even return to a previous state if stored. Emulators in general provide limited signal observability, although 100% visibility is possible at the cost of execution speed and hardware resources.

This section will introduce the commonly used hardware acceleration methods and the supporting technologies.

8.4.1 **Types of hardware acceleration**

Among the different methods to imitate a logic design, logic simulation and silicon implementation represent the two extremes in terms of resemblance to the final silicon. In between, various hardware acceleration techniques have been proposed to provide different balances among simulation speed, debugging capability, compilation time, and cost.

Figure 8.16a depicts a typical verification setup in which the "verification environment" provides the input vectors to the "design under verification" and analyzes the design's output responses. As illustrated in Figure 8.16b, in "simulation acceleration," the synthesizable part of the design under verification is mapped into hardware and executed in the emulator, whereas the remaining portions, in general the verification environment and the behavior code of the design, are executed in the workstation. High-speed channels exist between the workstation and the emulator to transport the simulation vectors and the responses. Because the simulator on the workstation is slower than the emulator, it becomes the bottleneck. However, with optimized testbench and simulation techniques, the high-speed channels could become the bottleneck.

In emulation (Figure 8.16c), the verification environment is also executed on the emulator to further increase the simulation speed. The communication between the workstation and the emulator is on demand, for example, to display the execution process. To do so, both the verification environment and the design itself must be synthesizable, implying a restricted coding style to the synthesizable subset.

In-circuit emulation (ICE) shown in Figure 8.16d relies on external hardware, which is usually the target system board, to provide "live-stimuli" and thus provides a more realistic verification environment. The target system may be

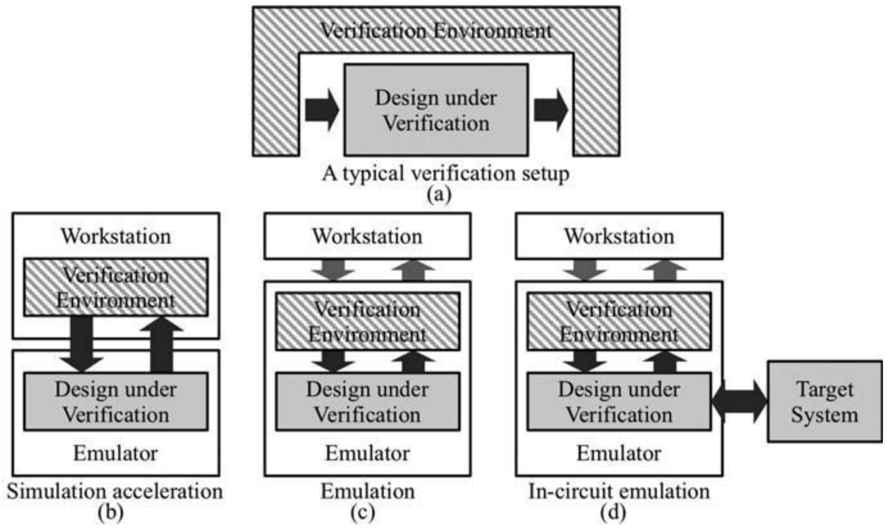


FIGURE 8.16

Simulation acceleration, emulation, and in-circuit emulation.

"static" or "dynamic." In the former case, the emulator supplies clock to the target system. Thus, one may stop and resume or slow down the simulation without causing problems. In the latter case, care must be taken to make sure that the whole system functions correctly (*e.g.*, to handle the differences in data and clock rate). As a result, the execution cannot be arbitrarily stopped, which limits debugging capability.

Hardware prototyping with FPGA is a well-known and widely adopted approach to verify the design correctness. The designers use the tool set provided by the FPGA vendors, including synthesis, placement, and routing, to map the design onto their FPGA technology. For small and medium-size ASIC's that can fit into a single FPGA, this approach is inexpensive and offers good simulation speed. However, as the number of FPGAs needed to realize the design increases, the partitioning and mapping process becomes cumbersome and error-prone.

The main ingredients of an emulator are the reconfigurable computing units programmed to perform the assigned tasks after design partitioning and the interconnection network that joins these computing units. They are discussed in the following sections.

8.4.2 Reconfigurable computing units

Today, the reconfigurable computing units for emulators are generally in the form of arrays of FPGAs or customized ASICs.

For an FPGA-based emulator, there are two FPGA use models. In one use model, the RTL design is partitioned into sub-circuits that can be fit into an FPGA. Then, each sub-circuit is synthesized at the gate level and mapped onto the target FPGA technology (Figure 8.17a). Because the gate count to I/O pin count ratios of the sub-circuits are very often greater than those of available commercial FPGAs, the FPGA logic is usually underused. As a result, the I/O pin resource becomes a severe limitation on this use model. Time-multiplexed interconnection schemes (Section 8.4.3.3) help relieve this I/O pin resource limitation.

In the second FPGA use model, the design is compiled into RCC (ReConfigurable Computing) elements [Lin 2002] (Figure 8.17b). Each RCC element is a small compact processor dedicated to perform one function (*e.g.*, Boolean expression, addition, multiplication, and case statement) at the RTL or gate level. This way, the design can be verified at different levels of abstraction, and the user does not have to debug at the gate level.

In [Beausoleil 1996], customized ASICs consisting of bit-wise Boolean processors are used as the computing units. The Boolean processors are designed to evaluate any *n*-input Boolean operations, and their inputs are selected with multiplexers. In this method, a circuit simulation cycle is divided into several emulator cycles, and the processors can perform different tasks and receive inputs from different sources in each emulator cycle according to the stored instructions. In Figure 8.18, the combinational network is simulated with three processors in 7 emulator cycles. The scheduling table shows one of the possible schedules. Processor 1, for example, samples *A*, *B*, and *C* in the first three emulator cycles and performs NOR(*A*, *B*, *C*) in the fourth emulator cycle.

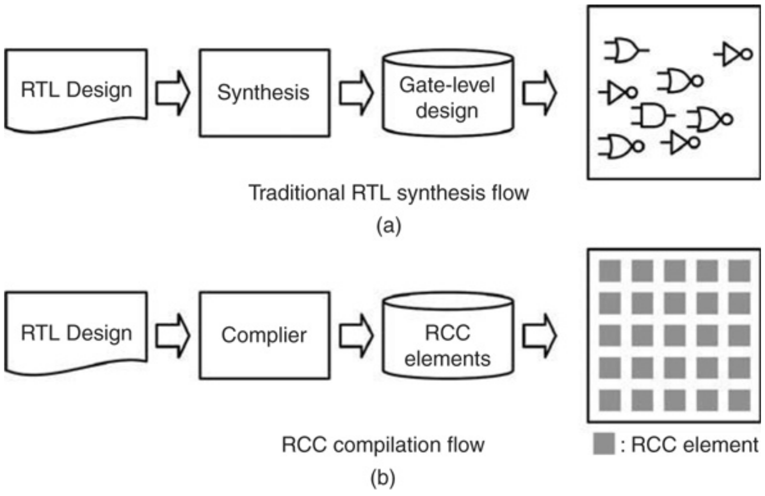
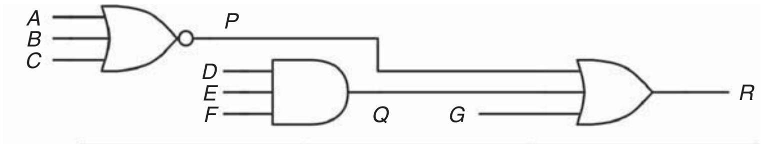


FIGURE 8.17

Traditional and RCC synthesis/compilation flows.



	Processor 1			Processor 2			Processor 3		
Cycle	I/P	Function	O/P	I/P	Function	O/P	I/P	Function	O/P
1	*A*			*D*			*G*		
2	*B*			*E*					
3	*C*			*F*					
4		NOR(*A*, *B*,*C*)	*P*		AND(*D*, *E*,*F*)	*Q*			
5							*P*		
6							*Q*		
7								OR(*G*,*P*, *Q*)	*R*

FIGURE 8.18

A processor scheduling example.

8.4.3 Interconnection architectures

The interconnection architectures can be divided into two categories: direct and indirect. In the former architecture, FPGA or ASIC chips are connected to each other directly through a fixed set of physical wires. In the latter architecture, dedicated routing chips are used to connect the chips, which relieves the computing units of inter-chip routing.

8.4.3.1 *Direct interconnection*

Because the computing units are usually arranged in a 2D array structure, the 2D-mesh-type direct interconnection architecture is an apparent choice. However, for the simple 2D-mesh interconnection scheme in Figure 8.19a, the valuable I/O pins may be used up by global interconnects routed through several computing units. In [Lin 2002], another mesh-type direct connection architecture (Figure 8.19b) is proposed. In this scheme, two "hops" and "jumps" are sufficient for any type of net; however, because each chip is used for routing as well as computing, the I/O pin resource limitation still exists.

In the direct interconnection architecture, the available I/O pin resource severely limits the logic use and the system routability. Because the chip I/O pin count grows at a slower rate than the gate count, either some logic resources must be used to route signals or some logic resources are wasted. Both indirect/time-multiplexed interconnection schemes and dynamically reconfigurable or programmable computing units help relieve this problem.

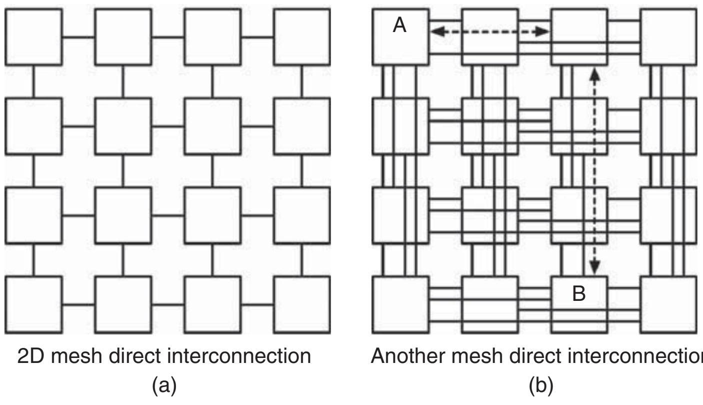

A
B
2D mesh direct interconnection
(a)
Another mesh direct interconnection
(b)

FIGURE 8.19

Direct interconnection architectures.

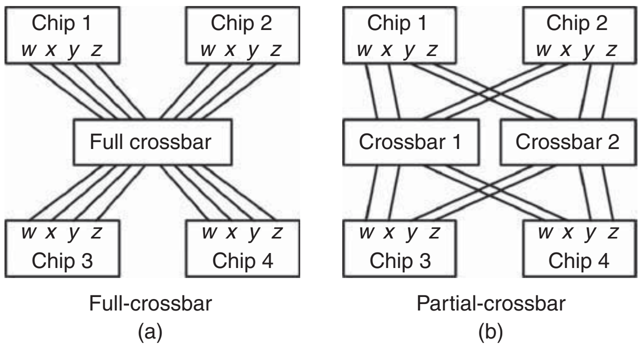


FIGURE 8.20

The full and partial-crossbar schemes.

8.4.3.2 ***Indirect interconnect***

The crossbar-based indirect interconnection scheme can be used to relieve the computing units of inter-chip routing. In the full-crossbar configuration (Figure 8.20a), one full-crossbar chip, after programming, is able to make connections between any two pins; however, this scheme soon runs out of steam, because the crossbar chip size increases quadratically with the total pin count. In the partial-crossbar configuration [Varghese 1993], the I/O pins are divided into groups, and one crossbar is assigned to connect pins of the same group. In Figure 8.20b, pins are divided into two groups (*w* and *x* in one group; *y* and *z* in the other), and two crossbars are used to realize intragroup connections. To further reduce the crossbar size, one may use the multilevel partial crossbar configuration.

8.4.3.3 *Time-multiplexed interconnect*

Alternatively, time-multiplexed interconnection schemes are used to overcome the pin resource limitation by dividing the pin bandwidth among several inter-chip logical signals.

The virtual wire technique in [Babb 1997] exploits the fact that logic values between circuit partitions only need to be transmitted once and that circuit communication patterns repeat in a predictable fashion.[1] As shown in Figure 8.21a, a physical wire is multiplexed among pipelined shift registers. Each register in the pipeline carries a single bit of information from one logical output to the corresponding logical input in the neighboring FPGA. Figure 8.21b illustrates the phase-based virtual wire operating principle. The simulation clock is divided into micro cycles; micro cycles are grouped into sequential phases to support combinational paths that extend across multiple chips. Each phase consists of the evaluation and communication time spans. In the former, logic outputs are evaluated according to the inputs; in the latter, the evaluation results are transferred to the other combinational logic partition.

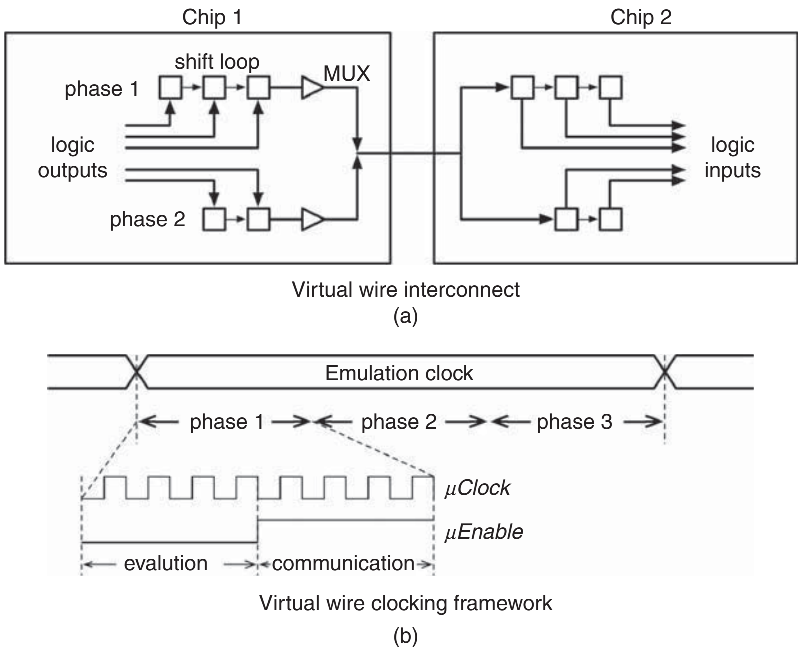


FIGURE 8.21

The virtual wire interconnection scheme.

[1]Assume that the circuit is synchronous and has no combinational loops.

In [Li 1998], a dynamic field programmable interconnect device (FPID), which consists of several layers of reprogrammable switching networks, is proposed. In Figure 8.22, there are *n* such reprogrammable switching networks, each of which can be a full or a partial crossbar. Here, *n* is the ratio of the speed of the physical wires and the switching network to that of the computing units. The *n* select lines are used to activate only one switching network at one time; thus, the I/O pin connections can be dynamically configured as expected.

The time-multiplexed interconnect scheme may be combined with the partial crossbar architecture as in Figure 8.23 [Sample 1999]. If *n*-to-1 multiplexers and de-multiplexers are used, the number of required pins can be reduced by a factor of *n*. In Figure 8.24, two signals share one I/O pin (*i.e.*, $n = 2$). Figure 8.24 depicts an example timing diagram for the wire *pq*. The *mux_clock* is divided by 2 to produce the divided clock *div_clock,* and the clock divider is reset by the *sync* signal to ensure that all clock dividers in the system are synchronized. *div_clock* is used to sample internal signal *p* at the rising edge and *q* at the falling

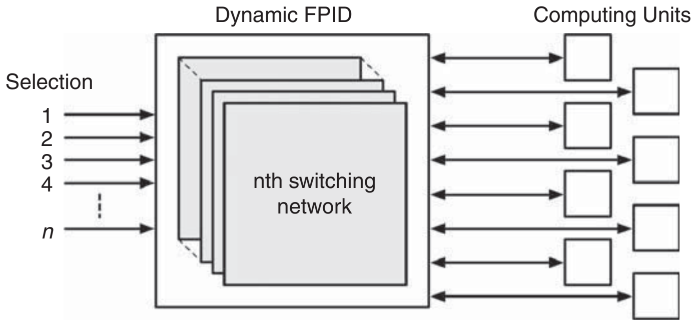


FIGURE 8.22

The dynamic FPID architecture.

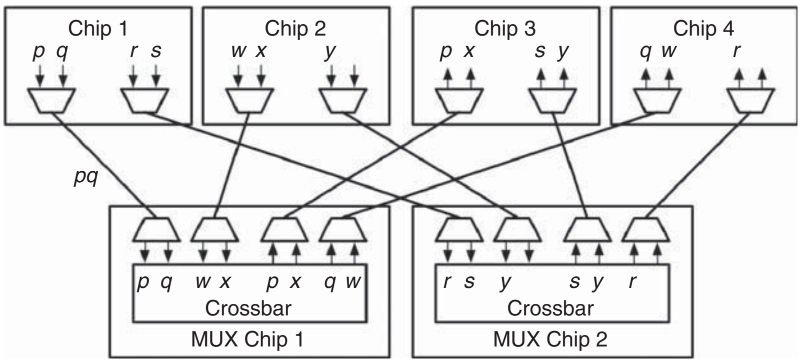


FIGURE 8.23

An interconnect scheme that combines TDI and partial crossbar.

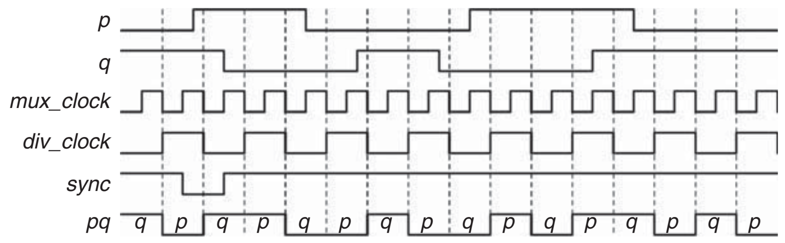


FIGURE 8.24

An example timing diagram of *pq* in Figure 8.23.

edge. Both *p* and *q* are stored in a flip-flop or a latch. When *div_clock* is high, *p* is transferred to the output; when *div_clock* is low, *q* is transferred to the output.

8.4.4 Timing issues

Achieving timing closure is a challenging task for emulators. Although many timing issues can be resolved by lowering the emulation frequency, this practice not only sacrifices the emulation throughput but also is insufficient for SOC designs that use multiple clock domains and different clock phases.

Consider the emulator architecture in Figure 8.15. The on-board clock generator supplies one or more clock signals to the FPGAs through PCB traces. These traces are carefully designed so that the clock edges arrive at the FPGAs concurrently; the clock trees inside each FPGA then forward these clock signals to the flip-flops and latches with as little skew as possible. However, things get complicated for internally generated secondary clocks. For example, the secondary clock signal *clk*_2 in Figure 8.25a is derived from the primary clock *clk*_1; it is generated in FPGA 1 but is also shared by FPGA 2. Because of the large inter-FPGA signal path delay, *clk*_2 signal can exhibit excessive skew. One solution to this problem is to duplicate the clock logic for each FPGA that requires *clk*_2 as shown in Figure 8.25b—each FPGA has its own copy of *clk*_2 (*i.e.*, *clk*_2$'$ and *clk*_2$''$, respectively).

Another timing issue is related to the hold-time error. In Figure 8.26, d_1 denotes the delay from *clk*_1 to D1's clock input, d_2 denotes the total delay from *clk*_1 to D2's clock input, and d_3 denotes the delay of the logic block. For flip-flop D2 to capture the correct logic block output, d_2 must be no less than the sum of d_1 and d_3. However, if d_2 gets too long, the next *clk*_1 edge may have arrived and cause the logic block output to change value before D2 can capture it; this leads to hold-time error. In an emulator, d_1, d_2, and d_3 depends on the emulator architecture; it is very difficult to adjust their values to ensure that hold-time errors do not occur. In [Wang 2006], the circuit to be emulated is modified so that the emulator can adequately control the clock edge timing relationship; correct operation of the modified circuit can be ensured by adjusting the clock frequency. The readers may refer to

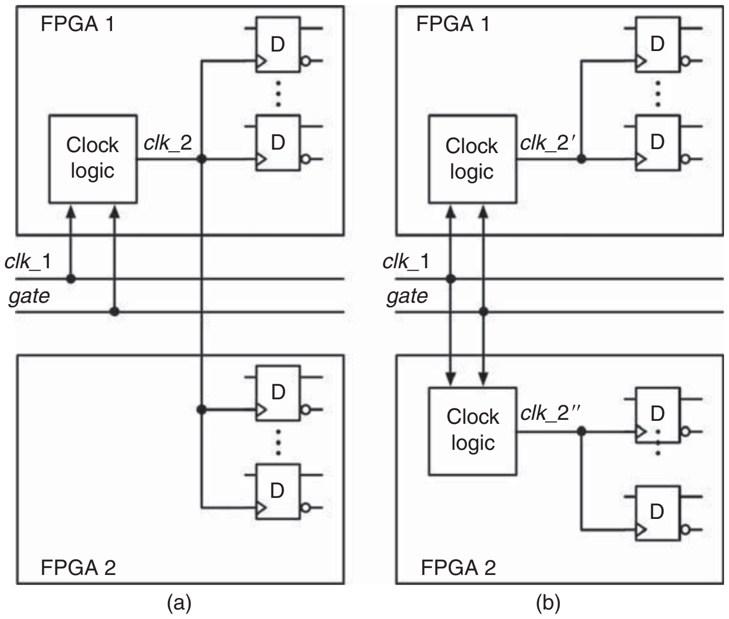


FIGURE 8.25

Duplicate clock logic to reduce clock skew.

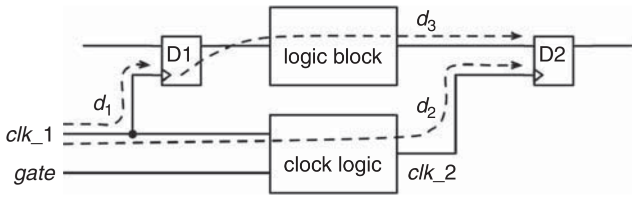


FIGURE 8.26

Hold-time error.

[Dai 1995], [Selvidge 1997], and [Tseng 2001] for other emulation techniques to eliminate hold-time errors.

8.5 CIRCUIT SIMULATION MODELS

In order to understand the modeling and simulation of ICs, we begin by visiting several basic circuit elements, namely, voltage and current sources, resistors, capacitors, and inductors. We also highlight several fundamental concepts from linear circuit analysis, specifically, the governing relationships for current and voltage throughout a circuit consisting of these basic elements. Next, we discuss the formulation of structured representations for circuit equations. In general, using analytical expressions for the solutions of these circuit equations would be computationally impractical for large-scale analysis. Hence, we will explore alternate numerical techniques for their solution.

8.5.1 Ideal voltage and current sources

An ideal voltage source is a circuit element where the voltage across the device is independent of the current flowing through it. There can be two kinds of ideal voltage sources.

- Independent voltage source: The voltage across the terminals is independent of any other variables in the circuit.
- Dependent voltage source: The voltage across the terminals is a function of some other variables in the circuit.

In a similar fashion, we can describe an ideal current source. An ideal current source is a circuit element where the current flowing through the device is independent of the voltage across it. Again, there are two kinds of ideal current sources.

- Independent current source: The current flowing through the device is independent of any other variables in the circuit.
- Dependent current source: The current flowing through the device is a function of some other variables in the circuit.

8.5.2 Resistors, capacitors, and inductors

Resistors, capacitors, and inductors are passive circuit elements. The amount of current flowing through a resistor is proportional to the voltage supplied across it. Ohm's law gives us the relationship between branch voltage (V), branch current (I), and resistance (R), which is measured in ohms (Ω):

$$V = IR$$

The charge Q stored on a capacitor is proportional to the voltage applied across it, and is given by

$$Q = CV$$

where C denotes the capacitance value measured in farads (F). Differentiating the preceding equation with respect to time t gives the following circuit equation for a capacitor:

$$I = C\frac{dV}{dt}$$

It is a well understood physical phenomenon that an electric current flowing through an inductor of inductance value L, which is measured in henrys (H), produces a magnetic flux Φ as follows:

$$\Phi = LI$$

Differentiating the preceding equation with respect to time gives the following circuit equation for an inductor:

$$V = L\frac{dI}{dt}$$

8.5.3 Kirchhoff's voltage and current laws

Kirchhoff's current law (KCL) states that at each node in a circuit, the sum of currents entering it is zero. This law is derived from the principle of conservation of charge. Consider the example circuit as shown in Figure 8.27. Applying KCL at node A gives $I_s - I_{l1} = 0$. Applying KCL at node B gives $I_{l1} - \mathrm{I}_{12} - I_{c1} = 0$. Applying KCL at node C gives $I_{l2} - I_{c2} = 0$.

Kirchhoff's voltage law (KVL) states that the voltage drop across every loop in a circuit is zero. This law is derived from the principle of conservation of energy. For the circuit shown in Figure 8.27, applying KVL on loop 1 gives $V_A - I_{l1}R_1 - L_1\ (dI_{l1}/dt) - V_B = 0$. Applying KVL across loop 2 gives $V_B - I_{l2}R_2 - L_2\ (dI_{l2}/dt) - V_C = 0$.

8.5.4 Modified nodal analysis

Through the combination of the branch equations and Kirchhoff's laws, governing equations can be systematically constructed to describe the dynamic behavior of the circuit. These formulations are often designed to exploit specific numerical methods for their solution. In addition, different formulations may be used in order to calculate quantities of interest. Specifically, the tableau formulation preserves all of the branch currents, branch voltages, and nodal voltages for the circuit. Alternatively, the nodal analysis (NA) formulation is used to solve only the nodal equations. Here, the nodal voltages are available during

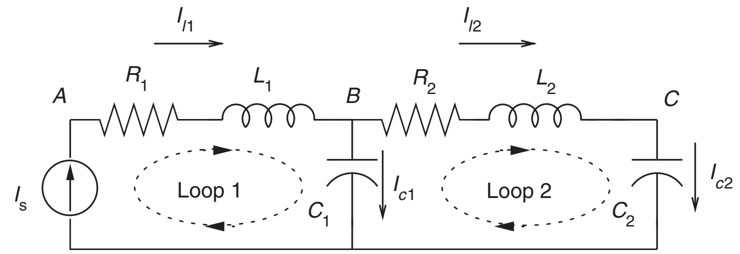


FIGURE 8.27

Example circuit.

simulation, which is the information most relevant to digital integrated circuit designers. However, the NA formulation cannot be used directly when the branch equations are dependent on branch currents (*e.g.*, when inductors and voltage sources are included). The ***modified nodal analysis*** (MNA) formulation also includes branch currents as unknown variables for these types of circuit elements [Ho 1975]. Due to the generality and computational efficiency of the MNA formulation, it is of broad interest to the circuit community, and we will focus on its development.

Given a linear circuit, we first define the adjacency matrix, A_J, for the system as follows:

$$A_J(i,j) = \begin{cases} +1 & \text{if node } j \text{ is the source of branch } i, \\ -1 & \text{if node } j \text{ is the sink of branch } i, \\ 0 & \text{otherwise.} \end{cases}$$

The adjacency matrix can be decomposed into sub-matrices according to the types of the branches:

$$A_J = \begin{pmatrix} A_i \\ \hline A_g \\ \hline A_c \\ \hline A_l \end{pmatrix}$$

Here, A_i, A_g, A_c, and A_l, respectively, correspond to the adjacency matrices for current sources, resistances, capacitances, and inductances. We also define the corresponding branch voltages and branch currents,

$$v_b = \begin{pmatrix} v_i \\ \hline v_g \\ \hline v_c \\ \hline v_l \end{pmatrix}, \quad i_b = \begin{pmatrix} i_i \\ \hline i_g \\ \hline i_c \\ \hline i_l \end{pmatrix}$$

which are related as follows:

$$i_i = I_s, \quad i_g = R^{-1} v_g, \quad i_c = C\dot{v}_c, \quad v_l = L\dot{i}_l$$

R is the resistance matrix and C is the capacitance matrix, both of which are diagonal. L is the inductance matrix. The corresponding adjacency matrices for R, L, and C are A_g, A_l, and A_c, respectively. I_s is the current source vector with adjacency matrix A_i. Here, we consider only current sources in the formulation due to the ability to convert voltage sources into their Norton equivalent circuits.

Now, we introduce v_n, the node voltages, which are related to v_g, v_c, and v_l through KVL as follows:

$$v_g = A_g v_n, \quad v_c = A_c v_n, \quad v_l = A_l v_n$$

Applying KCL, $A_J^T i_b = 0$ and eliminating most branch currents except for those flowing through inductors, we obtain the MNA formulation, which allows for a structured accumulation of current–voltage relationships in the form of matrix equations as follows:

$$\tilde{G}x + \tilde{C}\dot{x} = b \tag{8.1}$$

where

$$\tilde{G} = \begin{bmatrix} \mathcal{G} & A_l^T \\ -A_l & 0 \end{bmatrix}, \; \tilde{C} = \begin{bmatrix} \hat{C} & 0 \\ 0 & L \end{bmatrix}, \; x = \begin{bmatrix} v_n \\ i_l \end{bmatrix}$$

$$b = \begin{bmatrix} -A_i^T I_s \\ 0 \end{bmatrix}, \; \mathcal{G} = A_g^T R^{-1} A_g, \text{ and } \hat{C} = A_c^T C A_c$$

Example 8.1 For the circuit shown in Figure 8.27, the adjacency matrix is illustrated below:

$$A_J = \begin{pmatrix} A_i \\ \hline A_g \\ \hline A_c \\ \hline A_l \end{pmatrix} = \left(\begin{array}{cccccc} 1 & -1 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ \hline -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{array}\right)$$

The nodes, which correspond to the columns in A_J, are *GND*, *A*, *AB*, *B*, *BC*, and *C*, where nodes *AB* and *BC* are the nodes to the right of R_1 and R_2, respectively. As *GND* is the reference node in this example, we eliminate that column from A_J before constructing the matrices $\tilde{G}$ and $\tilde{C}$ as follows:

$$\tilde{G} = \begin{bmatrix} 1/R_1 & -1/R_1 & 0 & 0 & 0 & 0 & 0 \\ -1/R_1 & 1/R_1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1/R_2 & -1/R_2 & 0 & -1 & 0 \\ 0 & 0 & -1/R_2 & 1/R_2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}$$

$$\tilde{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & C_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & L_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & L_2 \end{bmatrix}$$

where

$$R = \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}, \quad L = \begin{bmatrix} L_1 & 0 \\ 0 & L_2 \end{bmatrix}, \quad C = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix}$$

The corresponding input vector *b* and the vector of voltage and current variables *x* are as follows:

$$b = [I_s \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$
$$x = [V_A \quad V_{AB} \quad V_B \quad V_{BC} \quad V_C \quad I_{l1} \quad I_{l2}]^T$$

Equation (8.1) can be separated into two smaller matrix differential equations as follows [Chen 2003]:

$$\begin{gathered}\mathcal{G}\, v_n + A_l^T i_l + \hat{C}\dot{v}_n = -A_i^T I_s \\ -A_l v_n + L\dot{i}_l = 0\end{gathered} \tag{8.2}$$

Although an analytic solution for Equation (8.1) or (8.2) exists, the evaluation of the solution is computationally expensive. Therefore, we shall explore numerical methods for solving ordinary differential equations of this type.

8.6 NUMERICAL METHODS FOR TRANSIENT ANALYSIS

Ordinary differential equations, or ODEs, are encountered when modeling the behavior of numerous physical systems. In order to construct a procedure for solving ODEs, we first have to understand general approximation methods for functions. This will serve as a foundation for the development of numerical techniques for integration. Finally, the application of these techniques to the solution of ODEs similar to the MNA Equations (8.1) and (8.2) will be addressed.

8.6.1 Approximation methods and numerical integration

First, we examine a general integration problem

$$I(f) = \int_a^b f(x)dx$$

where [*a, b*] is a closed and bounded interval. If no explicit form of the antiderivative exists or the evaluation time is prohibitively slow, the use of approximation methods becomes necessary. Let $\tilde{f}(x)$ be an approximation of $f(x)$ such that the approximation of our integral $I(\tilde{f}) = \int_a^b \tilde{f}(x)dx$ can be solved more readily or more efficiently.

In order to analyze how well $I(\tilde{f})$ can be used to approximate the original integral $I(f)$, we define the error for the estimate as follows:

$$\begin{aligned}|E(f)| &= \left|I(f) - I(\tilde{f})\right| \\ &= \left|\int_a^b \left[f(x) - \tilde{f}(x)\right]dx\right| \\ &\le \int_a^b \left|f(x) - \tilde{f}(x)\right|dx \\ &\le (b-a)\sup_{a\le x\le b}\left|f(x) - \tilde{f}(x)\right|\end{aligned}$$

In general, we would like the error to be as small as possible. In the following, we define a set of functions: $\{\tilde{f}_n | n \geq 1\}$ for the increasingly more accurate approximation of our integral, where

$$I_n(f) = \int_a^b \tilde{f}_n(x)dx = I(\tilde{f}_n)$$

We would like the error $|E_n(f)| = |I(f) - I_n(\tilde{f})| = \left|\int_a^b [f(x) - \tilde{f}_n(x)]dx\right|$ to diminish as n increases, that is, $\sup_{a \leq x \leq b}|f(x) - \tilde{f}_n(x)| \to 0$ as $n \to \infty$.

We choose $\{\tilde{f}_n | n \geq 1\}$ such that

$$I_n(f) = \sum_{j=0}^{n} w_{j,n} f(x_{j,n}) \tag{8.3}$$

where $w_{j,n}$ are called the weights (or quadrature weights) and $x_{j,n}$ are the integration nodes. For convenience of representation, we will leave out the n dependency.

If we choose to approximate f on $[a, b]$ by a straight line (see Figure 8.28) joining the points $(a, f(a))$ and $(b, f(b))$, then the single approximating function would be

$$\tilde{f}_1(x) = f(a) + \frac{(x-a)}{(b-a)}[f(b) - f(a)]$$

with the integral being of the form:

$$I_1(x) = \int_a^b \tilde{f}_1(x)dx = \left(\frac{b-a}{2}\right)[f(a) + f(b)]$$

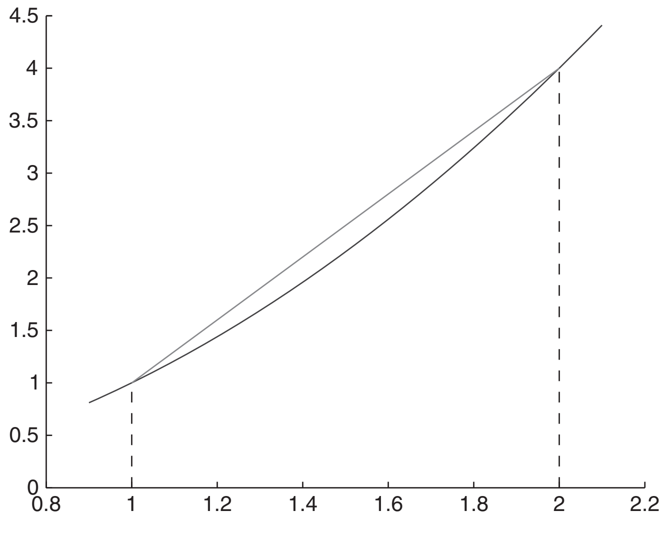


FIGURE 8.28

Approximation of the integral of $y = x^2$, $1 \leq x \leq 2$.

If we assume that f is twice differentiable, the error for our estimate is

$$E_1(f) = \frac{(b-a)^3}{12} f''(\eta), \quad \eta \in [a,b]$$

Example 8.2 $y = x^2$, $1 \leq x \leq 2$.

$$\begin{aligned} I(f) &= \int_1^2 x^2 dx = \frac{x^3}{3}\bigg|_1^2 \\ &= \frac{8}{3} - \frac{1}{3} \\ &= \frac{7}{3} \end{aligned} \qquad \begin{aligned} I_1(f) &= \left(\frac{2-1}{2}\right)[f(1) + f(2)] \\ &= \left(\frac{1}{2}\right)[1+4] \\ &= \frac{5}{2} \end{aligned}$$

$$\begin{aligned} E_1(f) &= -\frac{(2-1)^3}{12} f''(\eta) \\ &= -\frac{1}{12}(2) \\ &= -\frac{1}{6} \end{aligned} \qquad \begin{aligned} I(f) - I_1(f) &= \frac{7}{3} - \frac{5}{2} \\ &= \frac{14-15}{6} \\ &= -\frac{1}{6} = E_1(f) \end{aligned}$$

As $E_1(f) \propto (b-a)^3$, this estimate is inaccurate for large intervals. The general framework in Equation (8.3) allows us to break up the integral into smaller sub-regions as shown in Figure 8.29.

Specifically, we can form the *composite rule* as follows:

$$n \geq 1, \quad h = (b-a)/n, \quad x_j = a + jh, \quad j = 0, 1, \ldots, n$$

Assigning a linear function to approximate the function between the boundary points of each sub-region, we arrive at what is classically referred to as the *Trapezoidal Rule*:

$$\begin{aligned} I(f) &= \int_a^b f(x)\mathrm{d}x = \sum_{j=1}^{n} \int_{x_{j-1}}^{x_j} f(x)dx \\ &= \sum_{j=1}^{n} \left\{ \left(\frac{h}{2}\right)[f(x_{j-1}) + f(x_j)] - \frac{h^3}{12} f''(\eta_j) \right\}, \quad x_{j-1} \leq \eta_j \leq x_j \end{aligned} \tag{8.4}$$

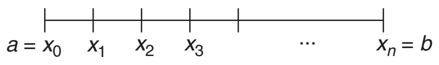


FIGURE 8.29

Defining subregions for the composite rule.

Therefore, we can write the approximating integral as:

$$I_n(f) = h\left[\frac{1}{2}f(x_0) + f(x_1) + \ldots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right]$$

with the error now quadratically dependent on the width of the sub-regions:

$$\begin{aligned} E_n(f) &= I(f) - I_n(f) \\ &= \sum_{j=1}^{n} -\frac{h^3}{12}f''(\eta_j) \\ &= -\frac{(b-a)h^2}{12}f''(\eta), a \le \eta \le b \end{aligned} \tag{8.5}$$

In order to further reduce the error, quadratic polynomials can be used to approximate the function, a method known as *Simpson's Rule*. Furthermore, the use of higher order approximations has been developed, and these approximations are called the *Newton-Cotes formulas* [Atkinson 1989].

8.6.2 **Initial value problems**

In general, we are interested in studying the initial value problem:

$$y' = f(x,y), \quad y(x_0) = Y_0 \tag{8.6}$$

where the function $f(x, y)$ is continuous for all (x, y) in some domain D, and Y_0 describes the initial condition for the differential equation. One of the simplest and most popular approaches to solving problems of this type is the *finite difference method*. Here we approximate the solution of Equation (8.6) at a discrete set of equally spaced grid points:

$$\begin{aligned} &x_0 < x_1 < x_2 < \ldots < x_n < \ldots \\ &x_j = x_0 + jh, \quad j = 0, 1, \ldots \end{aligned}$$

where the true solution $Y(x)$ satisfies Equation (8.6), and we will denote our approximation at each grid point $y_j = y(x_j), j \ge 0$. *Euler's method* gives us a first order method for relating future values of our approximate solution to past values, given the underlying differential equation and initial condition:

$$\begin{aligned} y_{n+1} &= y_n + hf(x_n, y_n), \quad n = 0, 1, \ldots \\ y_0 &= Y_0 \end{aligned}$$

Example 8.3 $y' = 3x^2 + x + 1, y_0 = 0.$

The solution is of the form:

$$Y(x) = x^3 + \frac{1}{2}x^2 + x + c$$

where c is a constant. Applying initial condition gives $Y(0) = 0^3 + \frac{1}{2}0^2 + 0 + c = c$. As $y_0 = 0$, we have $c = 0$. Therefore,

$$Y(x) = x^3 + \frac{1}{2}x^2 + x$$

On the other hand, Euler's method gives the following approximation:

$$y_{n+1} = y_n + h\left(3x_n^2 + x_n + 1\right)$$

The following table gives the errors $Y(x_j) - y_j$ across varying step sizes:

	***h* = 0.01**	***h* = 0.1**	***h* = 0.25**	***h* = 0.5**
***x* = 0.0**	0	0	0	0
***x* = 0.5**	–0.006275	–0.065	–0.171875	–0.375
***x* = 1.0**	–0.02005	–0.205	–0.53125	–1.125
***x* = 1.5**	–0.041325	–0.42	–1.078125	–2.25
***x* = 2.0**	–0.0701	–0.71	–1.8125	–3.75

Finally, we conclude by examining the relationship between numerical integration and the solution of ODEs. We first integrate Equation (8.6) across a region $[x_k, x_{k+1}]$ to obtain

$$Y(x_{k+1}) = Y(x_k) + \int_{x_k}^{x_{k+1}} f(x, Y(x))dx$$

Then, we apply Equation (8.4), the trapezoidal rule for integration, to reduce the preceding equation to

$$Y(x_{k+1}) = Y(x_k) + \frac{h}{2}[f(x_k, Y(x_k)) + f(x_{k+1}, Y(x_{k+1}))]$$
$$-\frac{h^3}{12}Y^{(3)}(\eta_j), \qquad x_k \le \eta_j \le x_{k+1}$$

which gives us the standard recursion for the trapezoidal method:

$$y_{k+1} = y_k + \frac{h}{2}[f(x_k, y_k) + f(x_{k+1}, y_{k+1})], \quad k \ge 0 \tag{8.7}$$

Consequently, the trapezoidal method has $O(h^2)$ convergence rate, which it inherits from the $O(h^2)$ error bound of the trapezoidal rule for integration in

Equation (8.5). It is also important to note that Equation (8.7) contains the unknown quantity y_{k+1} on both sides of the equality.

We can directly construct a numerical solution for the MNA circuit equations in Equation (8.1) by reformulating the problem in the form of Equation (8.6):

$$\begin{gathered} y' = \tilde{C}\dot{x} \\ f(t, x(t, y)) = -\tilde{G}x + b \\ x(t_0, y) = x_0 \end{gathered}$$

From here we can use the trapezoidal method in Equation (8.7), assuming a uniform discretization of the time axis with resolution h:

$$\begin{aligned} y_{k+1} &= \tilde{C}x_{k+1} \\ &= y_k + \frac{h}{2}[f(t_k, x(t_k, y)) + f(t_{k+1}, x(t_{k+1}, y))] \\ &= \tilde{C}x_k + \frac{h}{2}\left[\left(-\tilde{G}x_k + b_k\right) + \left(-\tilde{G}x_{k+1} + b_{k+1}\right)\right] \end{aligned}$$

Thus, we solve the following recursion for each time step of our simulation:

$$\left(\frac{\tilde{G}}{2} + \frac{\tilde{C}}{h}\right)x_{k+1} = -\left(\frac{\tilde{G}}{2} - \frac{\tilde{C}}{h}\right)x_k + \frac{b_{k+1} + b_k}{2} \tag{8.8}$$

8.7 SIMULATION OF VLSI INTERCONNECTS

On-chip interconnects, such as that shown in Figure 8.30, introduce capacitive, resistive, and inductive effects that can have a dominant impact on the circuit

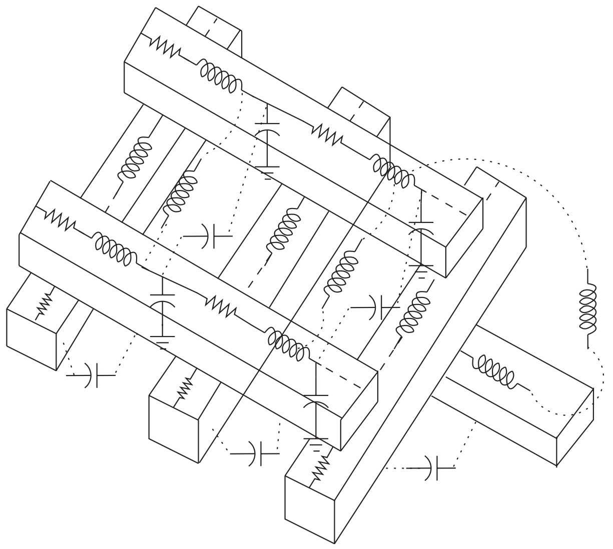


FIGURE 8.30

Distributed model of a typical three-dimensional VLSI interconnect structure.

operation and performance. In modern designs, the delay of global interconnects, possibly larger than the clock period, often dominates the gate delay. Moreover, coupling among interconnects can exacerbate the problem of signal integrity throughout the circuit. In this section, we will focus on the modeling and simulation of interconnects. The inclusion of nonlinear devices in a transient simulation procedure will be covered in the next section.

The capacitance of a wire is a typical component in nearly all interconnect models. The inclusion of resistive and inductive parameters in the modeling of interconnects is a recent trend. The scaling of devices reduces the resistance of a transistor. However, wire resistance increases significantly due to the scaling of the cross-sectional dimensions of interconnects and the increase in global interconnect lengths. As the relative significance of wire resistance increases, accurate models for the wire resistances must be employed. At higher frequencies, the impedance of an interconnect is mainly due to the inductance, which should be accounted for in a full-fledged interconnect model. In this section, we will examine a few commonly used models [Rabaey 1996, 2003; Cheng 1999], beginning with an analysis of the three physical quantities that are the building blocks for the models.

8.7.1 Wire resistance

The resistance of a rectangular wire with uniform cross-section, as shown in Figure 8.31, is given by:

$$R = \frac{\rho l}{wh} \tag{8.9}$$

where ρ is the resistivity of the material, and h, w, and l are, respectively, the height, width, and length of the wire. It is important to note that at lower frequencies, current flowing through a conductor can be assumed to be distributed uniformly throughout the cross-section of the conductor. Therefore, the resistance is accurately given by Equation (8.9). For higher frequencies, however, it is important to consider an electromagnetic induction phenomenon known as the *skin effect*. At high frequencies, current tends to crowd near the surface of the conductor, resulting in non-uniform distribution of current in a conductor, as shown in Figure 8.32, where a darker region indicates a higher

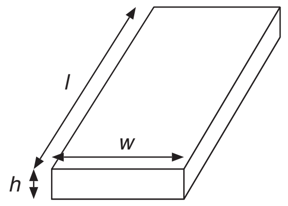


FIGURE 8.31

Rectangular wire.

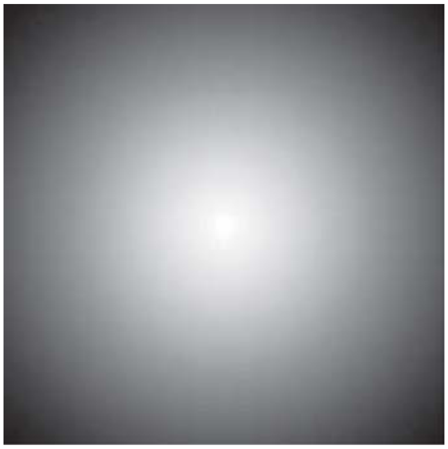


FIGURE 8.32

Non-uniform current distribution in a conductor caused by the skin effect.

current density. As most of the current is carried by a small portion of the cross-section of the conductor, the effective resistance increases.

To formalize this concept, we introduce the current density of a conductor as a function of depth *d* from the surface:

$$J = J_s e^{-d/\delta}$$

Skin depth δ is defined as the depth at which current density is attenuated to e^{-1} of its value at the surface:

$$\delta = \sqrt{\frac{2\rho}{\omega\mu}}$$

where $\omega = 2\pi f$ is the angular frequency and μ is the permeability of the conductor. It is important to note that for higher frequencies and higher conductivity values, there is less penetration of current into the conductor.

8.7.2 **Wire capacitance**

Traditionally, the capacitance of interconnects is the most influential parasitic parameter that the designer of a CMOS circuit has to consider, especially before the emergence of sub-micron technologies. Moreover, this methodology remains adequate when considering short interconnects that are not part of a critical path for the circuit.

In order to accurately model the behavior of interconnects we have to consider several capacitive components. Specifically, we examine *area*, *fringing*, and *coupling* capacitance components for the model. Both the area capacitance and the fringing capacitance are considered to be the *grounded* capacitance between the interconnect and the substrate. The coupling capacitance is the parasitic effect due to interactions between two neighboring interconnects. When compared to the grounded capacitance, we consider the coupling capacitance to be *floating*.

A simple capacitive model for a typical wire is given in Figure 8.33. The parallel plate capacitance is a result of the electrical field, shown normal to the surface of the conductor terminating at the ground plate, and is described by the following equation:

$$C = \frac{\varepsilon w l}{t_{\mathrm{ox}}}$$

where w and l are, respectively, the width and length of the wire, ε is the permitivity of the insulating material between the plates, and t_{ox} is the thickness of the insulator.

As the ratio w/h for the conductors decreases, which is typical when we scale the dimensions of interconnects, the parallel plate model becomes increasingly inaccurate. In this case, the capacitance between the side walls of the wires and the substrate (described by the fringing capacitance component) becomes a dominant contributor to the overall grounded capacitance. When approximating the total grounded capacitance, a typical approach treats the wires as rectangular sections with two hemi-spherical end caps, as described in [Yuan 1982] and illustrated in Figure 8.34. For this model, the total grounded capacitance is the sum of two components: a parallel-plate capacitance between a wire of width $w - h/2$ and the ground plane and a fringing capacitance modeled by a cylindrical wire with a radius of $h/2$. The interconnect grounded capacitance C_{grounded} can be calculated as follows:

$$C_{\mathrm{grounded}} = \varepsilon \cdot l \cdot \left[\frac{w - \frac{h}{2}}{t_{\mathrm{ox}}} + \frac{2\pi}{\ln \left\{ 1 + \frac{2t_{\mathrm{ox}}}{h} + \sqrt{\frac{2t_{\mathrm{ox}}}{h} \cdot \left(\frac{2t_{\mathrm{ox}}}{h} + 2 \right)} \right\}} \right]$$

The preceding discussion concerns the capacitance of a single interconnect. It is typical that every wire on a chip is surrounded by a number of neighboring

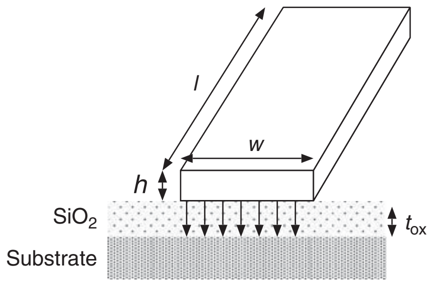


FIGURE 8.33

Parallel plate capacitance model of a wire.

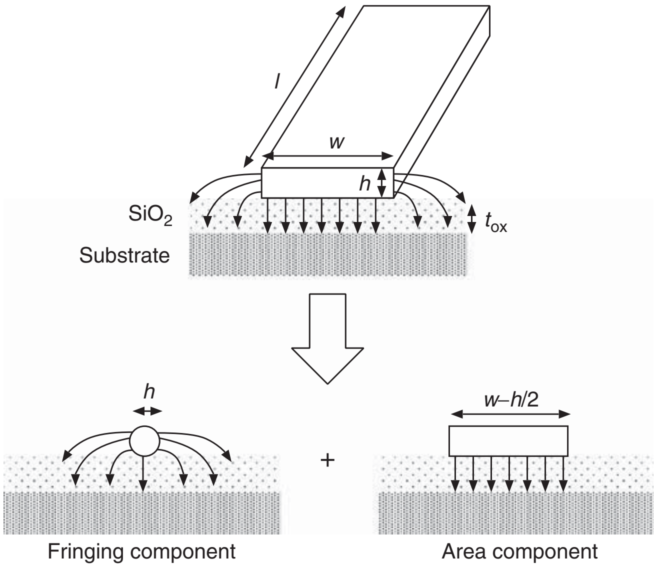


FIGURE 8.34

Fringing and area capacitance of a wire.

wires. These neighboring wires may completely or partially shield the wire of interest from the ground plane. As a result, the area capacitance and the fringing capacitance depend on the neighborhood configuration or the spacings from the neighboring wires as well.

While neighboring wires may reduce the grounded capacitance of a wire, they contribute to the coupling capacitances, which is due to electrical fields between adjacent wires that reside not only in the same layer, but also in different layers. Typically, three components of the coupling capacitance are considered. First, the area component of coupling capacitance is due to the parallel plates formed by the overlapping surfaces of wires in different routing layers. Second, the fringing component of the coupling capacitance is formed between the side-wall of one wire and the surface of a second wire above or below. Third, the lateral component of the coupling capacitance is due to the parallel plates formed by the side-walls of neighboring wires on the same layer. While there exist approximate models, fields-based solvers such as FastCap [Nabors 1991] are usually used to extract capacitive parameters of 3-D interconnect structures.

8.7.3 **Wire inductance**

With both the increase in clock frequencies and the decrease in signal transition times, on-chip inductance of interconnect wires has become a concern for circuit designers. The modeling of inductance effects is necessary when analyzing signal overshoot/undershoot and crosstalk noise. The self-inductance of a rectangular wire with uniform cross-section, as shown in Figure 8.31, can be approximated by the following equation [Keiser 1979]:

$$L = \frac{\mu_0}{2\pi}\left[l \ln\left(\frac{2l}{w+h}\right) + \frac{l}{2} + 0.2235(w+h)\right]$$

where μ_0 is the permeability of free space.

Exact formulas for computing the self-inductance and mutual inductance of rectangular wires are available in [Hoer 1965; Wu 1992; Zhong 2003]. As each of these formulas easily takes up more than half a page, we omit them in this textbook. These closed-form formulas are valid only at low frequencies, when the current distribution varies very little in the cross-sections and can be assumed to be uniform throughout the conductors. Consequently, the self and mutual inductances can be respectively computed one-by-one and pairwise even for a multi-conductor system.

At high frequencies, the current in a conductor is not uniformly distributed due to the skin effect. Moreover, the presence of neighboring wires also causes uneven distribution of current within a conductor, as shown in Figure 8.35. In this example, the currents in the two conductors flow in opposite directions. As current tends to flow in the path with the least loop impedance, the currents in the two conductors tend to crowd near the two closest surfaces of the conductors. This is known as the *proximity effect*. As the current distributions in the conductors affect each other, the inductive parameters of the whole system must be extracted at the same time, as is the extraction of capacitive parameters

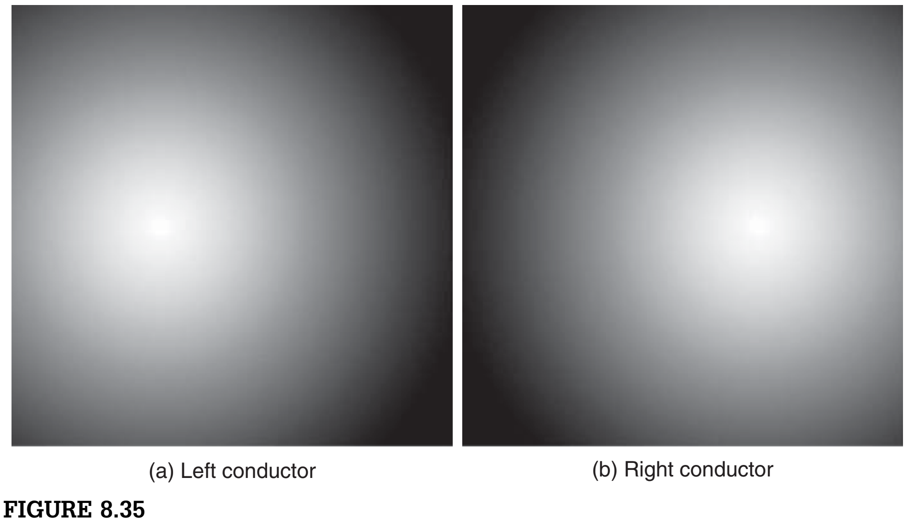


FIGURE 8.35

Non-uniform current distribution in two parallel conductors caused by the proximity effect.

of a multi-conductor system. The representative work is FastHenry [Kamon 1994], a parallel of FastCap [Nabors 1991].

8.7.4 **Lumped and distributed models**

The simplest model for an interconnect wire is a lumped capacitor model. As the resistive component of the wire becomes more significant, a ***resistive-capacitive*** (RC) model has to be adopted. The lumped RC model for a wire is shown in Figure 8.36. The use of a simple lumped RC model offers the potential to greatly simplify the analysis and optimization of interconnects, albeit with less accuracy.

The resistive and capacitive parasitics of a wire are in reality distributed along its length. In order to address the inaccuracy associated with a lumped model, especially when considering long interconnect wires, a distributed model can be formed by dividing a long wire into several segments, each of length ΔL. Let r and c be the unit-length wire resistance and capacitance, respectively. Each of the segments can be viewed as a lumped RC element with resistance $r\Delta L$ and capacitance $c\Delta L$. The distributed RC model for a wire is shown in Figure 8.37a.

In order to further improve upon the model, inductance can incorporated into the distributed framework; the distributed RLC line model is shown in Figure 8.37b, where l denotes the unit-length wire inductance. It is important to recall that in practice, there often exists significant coupling between parallel groups of interconnect wires. Therefore, in general we will consider both inductive and capacitive coupling between neighboring wire segments when we examine the dynamic behavior of the circuit as a whole. Consequently, the capacitance and inductance matrices are usually of large sizes. While capacitance matrices are in general sparse, the inductance matrices are dense.

8.7.5 **Simulation procedure for interconnects**

If we consider the structure of the MNA equations from Equation (8.2) in more detail, we arrive at the ***Nodal Analysis*** (NA) equations, used in [Chen 2003]. Specifically, using the trapezoidal method shown in Equation (8.7), we can formulate the following recursions:

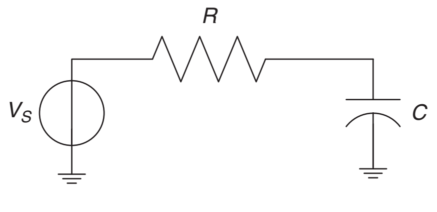
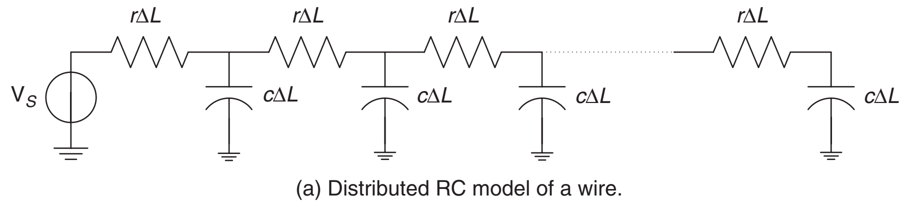
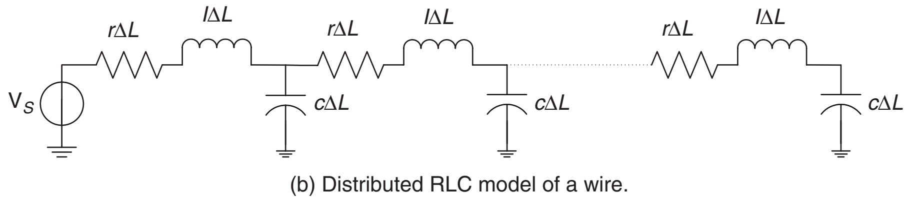


FIGURE 8.36

Lumped RC model of a wire.



(a) Distributed RC model of a wire.



(b) Distributed RLC model of a wire.

FIGURE 8.37

Distributed wire models.

$$\underbrace{\left(\mathcal{G}+\frac{2}{h}\hat{C}+\frac{h}{2}S\right)}_{U} v_n^{k+1} = \underbrace{\left(-\mathcal{G}+\frac{2}{h}\hat{C}-\frac{h}{2}S\right)}_{V} v_n^k - 2A_l^T i_l^k - A_l^T\left(I_s^{k+1}+I_s^k\right) \tag{8.10}$$

and

$$2A_l^T i_l^{k+1} = 2A_l^T i_l^k + hS\left(v_n^{k+1}+v_n^k\right) \tag{8.11}$$

where $S = A_l^T L^{-1} A_l$. In order to avoid overloaded subscripts for both node voltages and branch currents, the time steps are embedded in the superscripts of these variables in Equations (8.10) and (8.11). In contrast, we use subscripts to denote the time steps in Equation (8.8).

Equation (8.10) allows for the determination of the voltages v_n^{k+1} at each time step, which can then be used for the calculation of the currents i_l^{k+1} in Equation (8.11). The voltage and current variables can then be carried back to be used for solving the voltage equation in the next time step. The first advantage of the NA equations over the MNA representation is the reduced problem size. Separating the equations as shown in Equation (8.2) allows for the solution of a smaller pair of equations, which yields computational savings. In addition, the NA representation allows us to observe specific structures or sparsity in the matrices, thereby improving efficiency of simulation.

When large numbers of interconnects are modeled using a distributed framework, the inductance matrix L will be extremely large and dense (all entries are non-zero). The computation load involved in the simulation of even modestly sized problems using direct numerical techniques will be prohibitive. However, if we examine the inverse of the inductance matrix, we see specific structures that can potentially be exploited.

Consider, for example, a group of 16 parallel wires. Each row of the inductance matrix describes the mutual coupling between all of the conductors and a specific conductor (corresponding to the row being examined). In particular, each entry relates how the rate of change of current in an conductor, say *j*, contributes to the voltage of the conductor being examined, say *k*:

$$v_k = \sum_{j=1}^{16} L_{kj} \frac{di_j}{dt}$$

Now, let us take a look at the inverse relationship due to L^{-1}:

$$\frac{di_k}{dt} = \sum_{j=1}^{16} L_{kj}^{-1} v_j$$

Each entry in the inverse of the inductance matrix relates how the voltage change in a conductor affects the current in the conductor being examined. This is, in fact, what we are more concerned with in a digital circuit—how does the voltage switching in a conductor affect the signal delay or signal integrity of other conductors. From Figure 8.38 we can clearly see a substantial decrease in the magnitude of the entries for the first row of the inverse of the inductance matrix. This is due to the fact that the conductor number here corresponds directly to their spatial location, that is, the conductors are placed in rows beginning with the number with 1 and finishing with 16. Therefore, the farther away the conductors are from conductor 1, the less coupled they are. This is

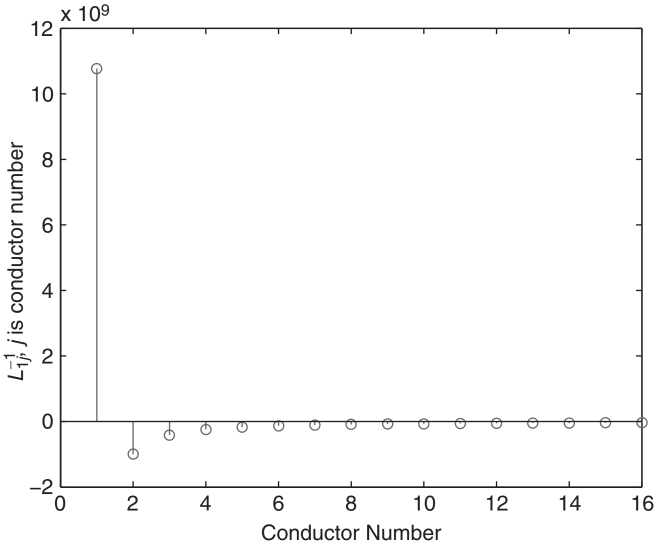


FIGURE 8.38

First row of the inverse of the inductance matrix for 16 parallel wires.

known as the locality effect or shielding effect. In other words, only conductors in close proximity of the conductor being examined are strongly coupled to the conductor of interest. These tightly coupled conductors shield faraway conductors from the conductor of interest.

This serves as the foundation for several approximation methods that reduce the amount of computation for simulation, while aiming to preserve accuracy. Several of these techniques rely on the fact that through the use of a threshold most of the entries in L^{-1} can be truncated (assumed to be identically zero), and all mathematical operations for the simulation can be performed with the resulting "sparse" L^{-1} matrices [Chen 2003]. Matrices U and V in Equation (8.10) are made up of sparse symmetric matrices $\mathcal{G}$, $\hat{C}$, and S. Cholesky factorization of sparse U can, therefore, be computed efficiently. The stored Cholesky factor of U can then be used for the repeated solves of v_n^{k+1}.

An alternate formulation that also allows us to exploit the inherent structure in matrices in the MNA equations can be constructed by rewriting the separated MNA Equation (8.2) as follows [Jain 2004]:

$$\begin{aligned} A_l i_l + \hat{C}_t v_n &= -A_i^T I_s, \\ -A_{gl} v_n + R i_l + L \dot{i}_l &= 0 \end{aligned} \tag{8.12}$$

where A_{gl} is the adjacency matrix formed by combining A_g and A_l and removing any zero columns created (from non-zero columns) as a result of inductor to resistor connections. Similarly, $\hat{C}_t$ is a truncated version of $\hat{C}$ obtained by removing the zero rows and columns in $\hat{C}$, which correspond to inductor-to-resistor connections. Typical VLSI interconnect has both resistance and inductance; this formulation relies on, as well as takes advantage of this property. For the example seen in Figure 8.27 we would have:

$$A_g + A_l = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \end{pmatrix}$$

As the third and fifth columns from $A_g + A_l$ are created from non-zero columns, we remove them to form A_{gl}:

$$A_{gl} = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix}$$

In this example, $\hat{C}_t = \hat{C}$.

Using the trapezoidal method shown in Equation (8.7) we can formulate the following recursions

$$\underbrace{\left(\frac{L}{h} + \frac{R}{2} + \frac{h}{4} A_{gl} P A_{gl}^T\right)}_{X} i_l^{k+1} = \underbrace{\left(\frac{L}{h} + \frac{R}{2} + \frac{h}{4} A_{gl} P A_{gl}^T\right)}_{Y} i_l^k + A_{gl} v_n^k + \frac{h}{4} A_{gl} P \left(I_s^{k+1} + I_s^k\right), \tag{8.13}$$

and

$$v_n^{k+1} = v_n^k - \frac{h}{2} P A_{gl}^T \left(i_l^{k+1} + i_l^k \right) + \frac{h}{2} P \left(I_s^{k+1} + I_s^k \right) \tag{8.14}$$

where P is the inverse of capacitance matrix, that is, $P = \hat{C}_t^{-1}$. The first equation allows for the determination of the currents i_l^{k+1} at each time step, which can then be used to solve for the voltages v_n^{k+1}. The voltage can then be carried back to be used for the next time step of the current equation.

The matrix X is dense as it is made up of dense matrices $P = \hat{C}_t^{-1}$ and L. As both $P^{-1} = \hat{C}_t$ and L^{-1} exhibit sparsity structure due to locality and shielding effects, X^{-1} can be expected to be sparse as well. Indeed, for a three-dimensional interconnect topology, we might find the significant entries for X^{-1} to have a pattern similar to that shown in Figure 8.39. Therefore, a truncated X^{-1} helps to reduce the computational complexity required of the repeated solves of i_l^{k+1}.

8.8 SIMULATION OF NONLINEAR DEVICES

Metal-oxide-semiconductor field-effect transistors (MOSFETs) and parasitic diodes constitute the basic circuit elements of modern digital circuits. In order to describe the dynamic behavior of a digital circuit, it is essential to construct

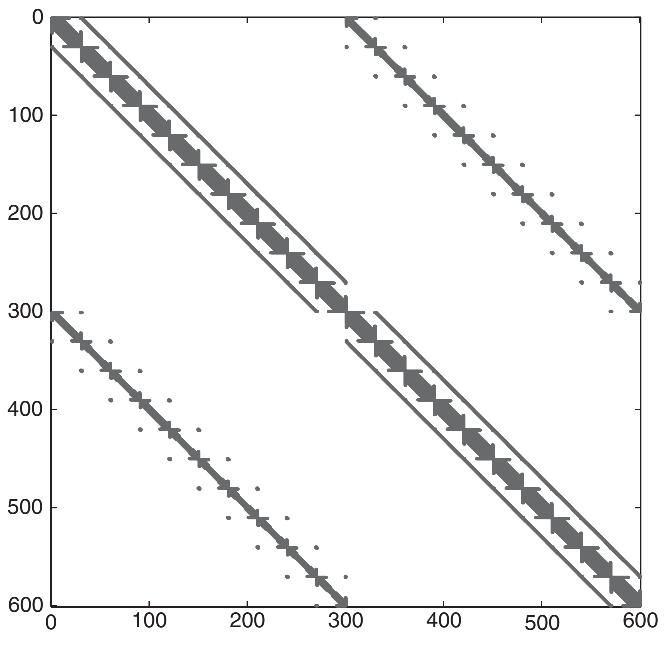


FIGURE 8.39

Significant entries for X^{-1} of three-dimensional interconnect structure.

models for these circuit building blocks. In general, the choice for a model is governed by both the accuracy in terms of describing the behavior of an actual circuit element in practice and the complexity (analysis time) associated with that model. Different models for devices exist based on the type of simulation performed by the designer. In this section, we limit our focus to device models used for transient simulation.

8.8.1 The diode

The diode, the schematic symbol of which is shown in Figure 8.40, is an important modern circuit element that is found in every MOSFET, the workhorse of modern digital circuits. Each source or drain diffusion region of a MOSFET naturally forms a *pn*-junction diode with the well in which the MOSFET resides.

The current-voltage characteristics or *I–V* characteristics for a typical diode can be divided into two regions: *forward-biased* and *reverse-biased*. When the voltage difference across the diode is smaller than a certain threshold voltage, it offers a very high resistance to the active current. Specifically, in the situation where no current is allowed to flow across the device we say that the diode is in the reverse-bias mode. As the potential drop is increased, the current can flow across the diode and the diode is said to be in the forward-bias mode. The *I–V* characteristics of a typical diode are shown in Figure 8.41. We can formulate the *I–V* characteristics of an ideal diode through the following equation:

$$I_D = I_S\left(e^{\frac{qV_D}{nkT}} - 1\right) \tag{8.15}$$

Here I_S is the reverse saturation current, q is the charge carried by an electron, k is Boltzmann's constant, T is the temperature in Kelvins, and n is the emission coefficient. The equivalent circuit model that provides us with Equation (8.15) is shown alongside the schematic symbol in Figure 8.40. The resistor R_s captures the series resistance due to the neutral regions on both sides of the junction. The nonlinear capacitance C_D, which is voltage-dependent, has two components, namely the *junction capacitance* and *diffusion capacitance*.

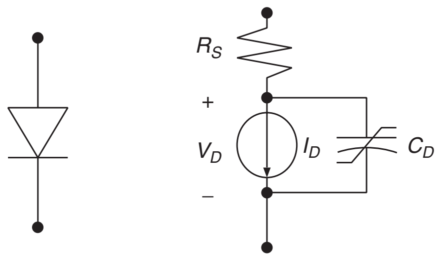


FIGURE 8.40

Circuit symbol for a diode and its equivalent circuit for transient analysis.

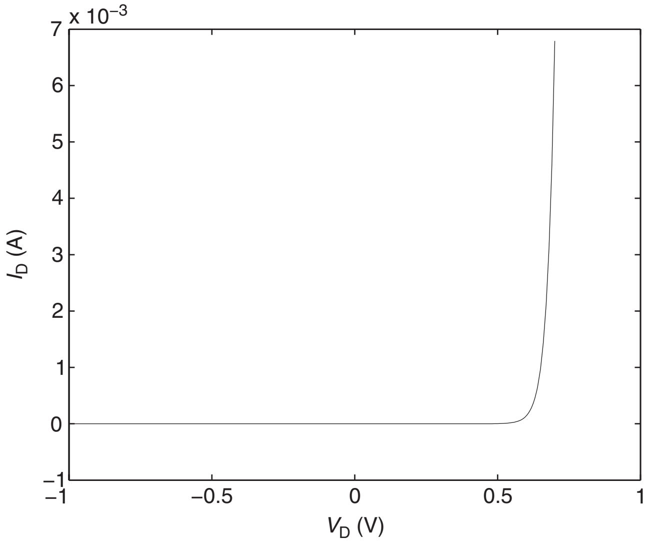


FIGURE 8.41

I–V characteristics of a diode.

The charge variation in the depletion region of a *pn*-junction diode due to variation in the potential difference across the junction is modeled as a nonlinear junction capacitance. The junction capacitance can be seen as the parallel-plate capacitance between the *n*-regions and *p*-regions of a *pn*-junction diode, and is given by the following expression [Rabaey 1996]:

$$C_j = \frac{dQ_j}{dV_D} = A_D \sqrt{\frac{q\varepsilon_{si}}{2} \frac{N_A N_D}{N_A + N_D} (\phi_0 - V_D)^{-1}}$$

where A_D is the junction area, ε_{si} is the permittivity of silicon, ϕ_0 is the zero bias potential across the junction, and N_A and N_D are the acceptor and donor concentrations, respectively. The preceding equation is valid only for an abrupt junction that has an instantaneous transition from *p*-material to *n*-material. For a linearly graded junction, a variation of the preceding equation can be used to model the junction capacitance [Rabaey 1996].

Under forward bias, excess carriers are stored at the boundaries of the depletion region. This effect is modeled by the diffusion capacitance, which is approximated by the following expression [Rabaey 1996]:

$$C_d = \frac{dQ_d}{dV_D} = \frac{q\tau_T I_S}{k\mathrm{T}} e^{qV_D/nkT}$$

where τ_T is the mean transit time for the charge to flow across the diode.

8.8.2 The field-effect transistor

The metal-oxide-semiconductor field-effect-transistor (MOSFET) is the key component in present-day VLSI circuits. There are several existing models with varying degrees of sophistication that have been presented in literature [Tsividis 1987]. In this section, we will concentrate on an NMOS transistor, whose schematic symbol is shown in Figure 8.42. The behavior of a MOS transistor can be separated into three modes of operation depending on the voltages applied across its terminals: gate (G), source (S), drain (D), as well as body (B) or bulk. For simplicity, we assume that the body is tied to ground. The *I–V* characteristics of a *long-channel* NMOS transistor for each of the three modes can be described by the following equations:

- Cutoff Region ($V_{GS} \leq V_T$), where V_T is the threshold voltage of the transistor:

$$I_{DS} = 0$$

- Linear Region ($V_{DS} < V_{GS} - V_T$):

$$I_{DS} = \mu_n C_{ox} \frac{W}{L}\left((V_{GS} - V_T)V_{DS} - \frac{V_{DS}^2}{2}\right)$$

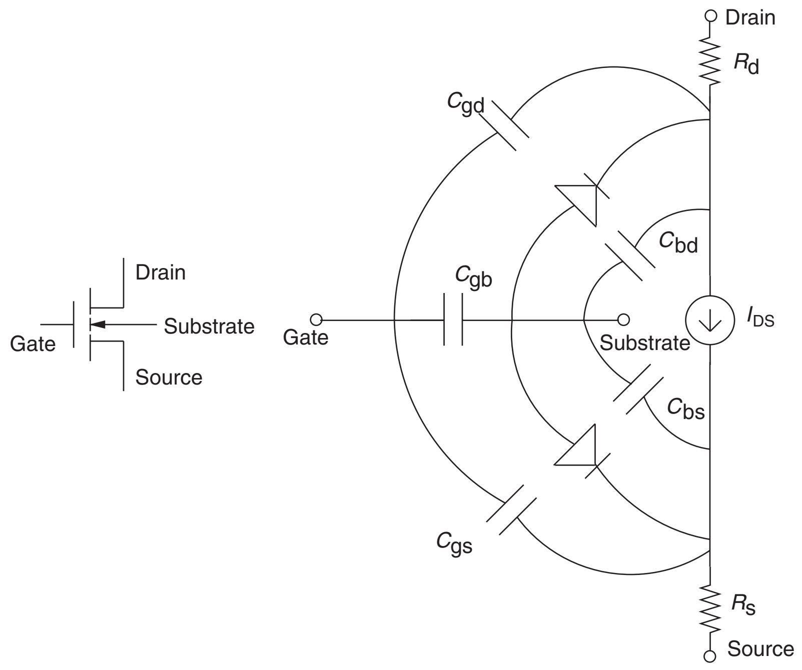


FIGURE 8.42

Circuit symbol of an NMOS transistor and its equivalent circuit for transient analysis.

Here, μ_n is the mobility of the transistor, C_{ox} is the per unit area capacitance of the oxide, and W and L are, respectively, the width and length of the transistor.

- Saturation Region ($V_{\text{DS}} \geq V_{\text{GS}} - V_{\text{T}}$)

$$I_{\text{DS}} = \mu_n C_{\text{ox}} \frac{W}{L} \frac{(V_{\text{GS}} - V_T)^2}{2}$$

In the preceding equation, we assume that in saturation mode the transistor will act like a perfect current source. However, the applied voltage at the drain would shorten the channel length. To account for that, we must include some dependence of the actual effective channel length on V_{DS}. This is accomplished through the inclusion of a channel-length modulation factor λ:

$$I_{\text{DS}} = \mu_n C_{\text{ox}} \frac{W}{L} \frac{(V_{\text{GS}} - V_T)^2}{2} (1 + \lambda V_{\text{DS}})$$

The *I–V* characteristics for a typical long-channel NMOS transistor are shown in Figure 8.43a.

Short-channel devices exhibit current-voltage characteristics that are considerably different from the long-channel devices. In particular, we have to incorporate the velocity-saturation effect of carriers in the equations for I_{D}. In a long-channel device, we assume that the velocity of carriers is proportional to the electrical field. For a short-channel device, the velocity remains constant or saturates at υ_{sat} when the electrical field reaches a critical value, ξ_{c}. An in-depth analysis of this effect is outside the scope of this book. Interested readers are encouraged to explore [Rabaey 2003]. For short-channel devices, the equation for I_{D} in the linear region is

$$I_{\text{DS}} = \mu_n C_{\text{ox}} \frac{W}{L} \left((V_{\text{GS}} - V_T) V_{\text{DS}} - \frac{V_{\text{DS}}^2}{2} \right) \kappa(V_{\text{DS}})$$

where

$$\kappa(V) = \frac{1}{1 + V/(\xi_{\text{c}} L)}$$

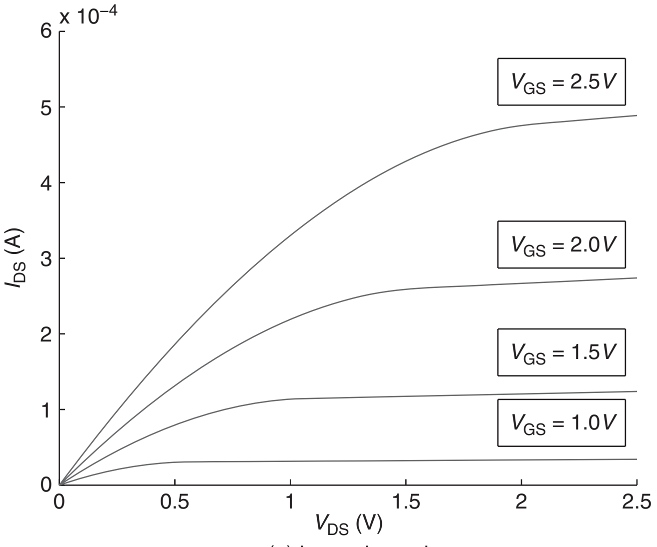
When the velocity of carriers saturates, the transistor operates in the saturation region, and the equation for I_{D} becomes

$$I_{\text{DS}} = \upsilon_{\text{sat}} C_{\text{ox}} W (V_{\text{GS}} - V_T - V_{\text{DSAT}})$$

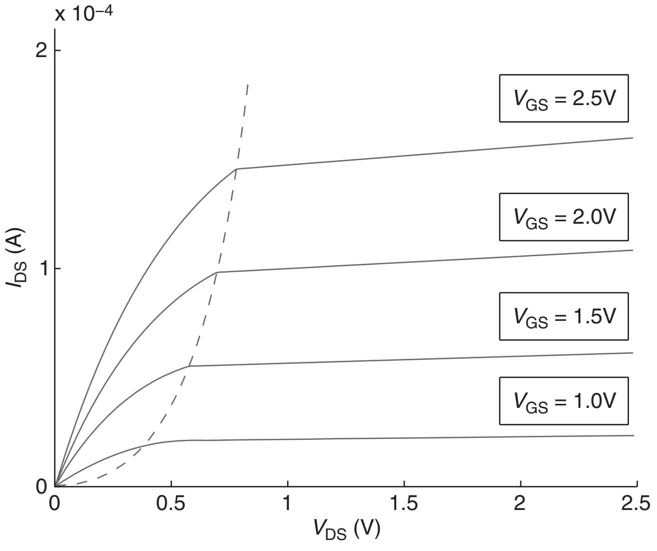
where $V_{\text{DSAT}} = (V_{\text{GS}} - V_{\text{T}})\kappa(V_{\text{GS}} - V_{\text{T}})$ is the drain-source voltage at which velocity saturation occurs. Again, the accuracy of the model can be further improved by including the channel length modulation factor. The *I–V* characteristics for a short-channel NMOS transistor is shown in Figure 8.43b.

While it is fine to assume $I_{\text{DS}} = 0$ when $V_{\text{GS}} \leq V_{\text{T}}$ for long-channel devices, sub-threshold leakage is no longer negligible for short-channel devices. The current in the sub-threshold region can be approximated as follows [Rabaey 2003]:

$$I_{\text{DS}} = I_S e^{\frac{qV_{\text{GS}}}{nkT}} \left(1 - e^{-\frac{qV_{\text{DS}}}{kT}} \right) (1 + \lambda V_{\text{DS}})$$



(a) Long-channel



(b) Short-channel

FIGURE 8.43

I–V characteristics of an NMOS transistor.

where I_S and n are empirical parameters, with $n \approx 1.5$.

We now describe the different capacitances associated with an MOS transistor. First, due to lateral diffusion under the poly gate during the ion implanation of source and drain, the gate overlaps with source and drain by x_d. Consequently, a transistor of gate length L has an effective length of $L_{eff} = L - 2x_d$. The overlap capacitance at the source or drain is $C_{ox}Wx_d$, where W denotes the channel width of the transistor. The gate-to-channel capacitance of a MOSFET can be divided into three capacitances, with C_{gs}, C_{gb}, C_{gb} being the capacitances between the gate and source, drain, and body respectively. These capacitances vary as the operating condition varies. An approximation of these capacitance values are given in Table 8.5.

Alternatively, a more accurate piecewise-linear approximation is typically used to make these capacitances continuous, resulting in a more stable simulation. For example, we can represent the gate-to-drain capacitance as:

$$C_{gd} = -\frac{dQ_g}{dV_D} \tag{8.16}$$

where the rate of change for both the gate charge Q_g and drain voltage V_D are obtained from the estimated increases over a fixed period of time. While it should be clear that the drain voltage can be determined directly from KVL-type equations, there are several different models available to approximate the charges seen at different areas of the MOSFET. If we consider the BSIM4 model [Dunga 2007], which is often provided as part of commercially available simulation packages, the amounts of charge observed at different terminals (*i.e.*, gate, body, drain and source) of a MOSFET in the saturation region will be as follows:

$$\begin{aligned}
Q_g &= C_{oxe}W_{active}L_{active}\left[V_{gs} - V_{FB} - \Phi_s - \frac{V_{DSAT}}{3}\right] \\
Q_b &= -C_{oxe}W_{active}L_{active}\left[V_{FB} + \Phi_s - V_T + \frac{(1 - A'_{bulk})V_{DSAT}}{3}\right] \\
Q_d &= -\frac{4}{15}C_{oxe}W_{active}L_{active}(V_{GS} - V_T) \\
Q_s &= -(Q_g + Q_d + Q_b)
\end{aligned} \tag{8.17}$$

Table 8.5 Approximation of gate capacitances

Mode	C_{gs}	C_{gd}	C_{gb}
Cutoff	$C_{ox}WL_{eff}$	0	0
Linear	0	$\frac{C_{ox}WL_{eff}}{2}$	$\frac{C_{ox}WL_{eff}}{2}$
Saturation	0	$\frac{2C_{ox}WL_{eff}}{3}$	0

The model shown above is dependent on several parameters such as the effective gate oxide capacitance, C_{oxe}, the effective length and width of transistor, L_{active} and W_{active}, the surface potential Φ_{s}, the flat-band voltage V_{FB}, as well as A'_{bulk}, a parameter related to the bulk charge effect. Default values for these and other parameters can be found in resources similar to [Dunga 2007], along with a complete description of the charge relationships in all operating regions for the device.

The other two capacitances C_{bs} and C_{bd} shown in Figure 8.42 are the diffusion capacitances, contributed by the reverse biased source-bulk and drain-bulk *pn* junctions, respectively. They can be modeled by the nonlinear capacitance C_D associated with a *pn*-junction diode, as described in Section 8.8.1.

8.8.3 Simulation procedure for nonlinear devices

It is often the case that simulation problems involve nonlinear circuit device components. In this context, the nonlinearity refers to the nonlinear current-voltage relationship over time. Many nonlinear devices can be described using the simple building blocks of voltage/current sources, resistors, capacitors, and inductors. It is important to note that both the values of the voltage/current sources and the parasitic values for the model may change with time. This is due to the fact that the dynamic behavior for nonlinear devices is governed by the voltages seen at the terminals of the device. For example, if we consider the model of the MOSFET it should be clear that the drain current I_{DS} is dependent on the voltages seen at terminals, for example, V_{GS} and V_{DS}. In addition, from Figure 8.42 we can see that the capacitance values between the terminals, for example, C_{gd} and C_{gs}, would again be dependent on the potential difference seen across the terminals. Therefore, we should consider a simple "black box" approach to analyzing these nonlinear devices. Here, we will assume for our simulation a device that is dependent on both the current and voltage values seen at the terminals over time.

For example, the current seen at the output terminal can be described by some nonlinear function g of the voltage seen at each of the terminals across time:

$$i_{\mathrm{out}} = g(v_{\mathrm{in}}, v_{\mathrm{out}}, t)$$

If we place this black box inside a simple circuit consisting of a voltage source and a load capacitance, shown in Figure 8.44, we can write down the voltage current relationships. For simplicity of illustration we will consider the nonlinear device to be the diode described in Section 8.8.1. Moreover, we ignore the nonlinear capacitance C_{D} associated with the diode. Here the current across the diode is given as:

$$i_{\mathrm{D}} = I_{\mathrm{S}}\left[e^{\frac{q}{nkT}(v_{\mathrm{s}}-v_1)} - 1\right]$$

and the current across the capacitor is:

$$i_C = C\frac{dv_1}{dt}$$

Combining the equations we get the ODE:

$$v_1' = \frac{I_s}{C}\left[e^{\frac{q}{nkT}(v_s - v_1)} - 1\right] = g(t, v_1)$$

Using the integration technique described in Section 8.6.2, we are left with a nonlinear expression

$$v_1^{k+1} = v_1^k + \frac{h}{2}\left[g(t_k, v_1^k) + g(t_{k+1}, v_1^{k+1})\right]$$

or

$$v_1^{k+1} - v_1^k - \frac{h}{2}\left[g(t_k, v_1^k) + g(t_{k+1}, v_1^{k+1})\right] = 0 \tag{8.18}$$

In this formulation, we know the value of v_1^k and we are interested in solving for the value of v_1^{k+1}, that is, the voltage at the next instant in time. There are several classical methods that can be used to find the "zero" of this nonlinear equation. We will focus on Newton's method, which offers a computationally efficient and numerically stable approach to solve for a zero. If we denote the unknown variable to be $x = v_1^{k+1}$, and the left-hand side of Equation (8.18) to be $z(x)$, then we are attempting to determine a solution x^* such that our nonlinear equation $z(x^*) = 0$. This is accomplished by forming a sequence of iterates that will converge to this point x^*, starting with an initial guess x_0:

$$x_{n+1} = x_n - \frac{z(x_n)}{z'(x_n)}, \quad n = 0, 1, \ldots \tag{8.19}$$

where $x_n \rightarrow x^*$ as $n \rightarrow \infty$. Note here that the subscripts in the preceding equation refer to the iteration numbers of the iterative procedure. In general, this procedure can be followed exactly when dealing with the inclusion of nonlinear devices into the MNA framework. Specifically, the effect of the nonlinear devices can be explicitly seen in the variable b below:

$$\tilde{G}x + \tilde{C}\dot{x} = b \tag{8.20}$$

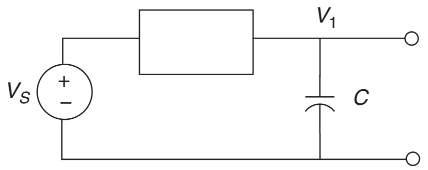


FIGURE 8.44

Circuit example including nonlinear device model.

where

$$\tilde{G} = \begin{bmatrix} \mathcal{G} & A_l^T \\ -A_l & 0 \end{bmatrix}, \; \tilde{C} = \begin{bmatrix} \hat{C} & 0 \\ 0 & L \end{bmatrix}, \; x = \begin{bmatrix} v_n \\ i_l \end{bmatrix}$$
$$b = \begin{bmatrix} -A_i^T I_s + I_{\text{nl}} \\ 0 \end{bmatrix}, \; \mathcal{G} = A_g^T R^{-1} A_g, \text{ and } \hat{C} = A_c^T C A_c$$

The vector I_{nl} captures the effect of nonlinear loads and depends on the node voltages as $I_{\text{nl}} = f(v_n)$. Here, f is a function that varies depending on the load characteristics and in general, can be a nonlinear function. In addition, it is important to note that the values of the capacitance matrix C will change as the simulation progresses. This can clearly be seen by revisiting the piece-wise-linear approximations which were used for the junction capacitances in Equation (8.16). As the node voltages v_n change with time, we will see a corresponding change in the capacitance values reflected by changes to the matrices seen in Equation (8.20).

The result is a voltage-dependent system of linear equations that must be solved at each time step of the simulation process. At first glance, this would numerically lend itself to the use of an iterative solver. We can, for example, apply Equation (8.19) iteratively to solve for each x^k at the kth time step. This is due to the fact any preprocessing or initial factorization would not directly be of benefit from one time step to the next as the matrices change over time. However, the matrix equations observed during the trapezoidal integration and Newton's method steps described above have a very special structure when combining nonlinear devices with various interconnect topologies. Specifically, the matrices can be grouped into those that remain constant with respect to time, and those components that are time-dependent or voltage-dependent.

Based upon a nodal analysis scheme, it should be clear that these two sets are not independent, that is, there exist branches or coupling between the nodes, and they must be eventually solved together. However, the special nature of the underlying matrix elements lends itself to the use of different numerical techniques for efficient solution. Specifically, the linear portion is time-independent and a direct factorization (*e.g.*, LU or Cholesky) can be reused for each time-step. The nonlinear, time-dependent portion will not benefit from this preprocessing; however, a preconditioned iterative method can be employed (see [Jain 2006] and [Zhu 2007], for details relating to this procedure).

8.9 CONCLUDING REMARKS

We have presented two classes of fundamental techniques, logic simulation and circuit simulation, which are useful for predicting the behavior of a design before its physical implementation is built. During this design stage, designers heavily rely on simulation to verify and debug their designs. Oftentimes, simulation is combined with functional verification, which is a major topic in Chapter 9, to further ensure the correctness of their designs. Commercial simulation tools have been available for modern ***system-on-chip*** (SOC) designs modeled at the behavioral level down to the lowest transistor level. This chapter covered the fundamental logic simulation, hardware-accelerated logic simulation, and circuit simulation techniques at the gate level and the transistor level.

For logic simulation, event-driven simulation that can take timing (delay) models and sequential circuit behavior into consideration is the technique most widely used in commercially available logic simulators. Examples of logic simulators include Verilog-XL, NC-Verilog (both from Cadence [Cadence 2008]), ModelSim (from Mentor Graphics [Mentor 2008]), and VCS (from Synopsys [Synopsys 2008]). These logic simulators can accept gate-level models as well as RTL and behavioral descriptions of the circuits written in hardware description languages, such as Verilog and VHDL, both of which are IEEE standards. HDLs are beyond the scope of this book but are important topics for digital designers to learn. More detailed descriptions of both languages can be found in books or Web sites, such as [Palnitkar 1996], http://www.verilog.com, and http://www.verilog.net.

Although flexible and low-cost, software simulators are becoming too slow for modern SOC designs and hardware/software co-simulation applications. Hardware-accelerated logic simulation techniques have been developed to bridge the growing gap. Emulators are differentiated by their interconnection architectures and the types and use models of the reconfigurable computing units. Each combination has its advantages and disadvantages and finds its applications in different verification environments. A few popular commercially available emulators include Incisive Acceleration and Emulation (from Cadence [Cadence 2008]), ZeBu (from EVE [EVE 2008]), and Veloce (from Mentor Graphics [Mentor 2008]).

Circuit simulation (commonly referred to as SPICE simulation [Nagel 1975]) at the transistor level, although too slow for practical designs, is important when the circuit's dynamic behavior or accurate timing information is desired. In general, circuit simulation is used to characterize the cell library, memory models, and the timing critical portion of the circuit. A few popular circuit simulators include Hspice (from Synopsys [Synopsys 2008]) and Spectre (from Cadence [Cadence 2008]).

As the design complexity continues growing and has reached two billion transistors [Stackhouse 2008], verifying the correctness of these designs has become a much more challenging task than ever. As a result, it is becoming imperative that advanced techniques for both logic simulation and circuit simulation, either hardware-accelerated or pure software-based, be developed to address the high-performance and high-capacity issues.

8.10 EXERCISES

8.1. (3-Valued Logic Simulation) By use of 3-valued logic simulation, what is the output value of circuit *M* in Figure 8.45 if the input pattern is $ABCD = 1u0u$? Show that there is information loss in this example.

8.2. (Timing Models) For circuit *M* shown in Figure 8.45, complete the timing diagram in Figure 8.46 with respect to each timing model given below:

a. *Nominal delay*—Two-input gate, 10 ns; three-input gate, 12 ns; inverter, 8 ns.
Inertial delay—All gates, 4 ns.

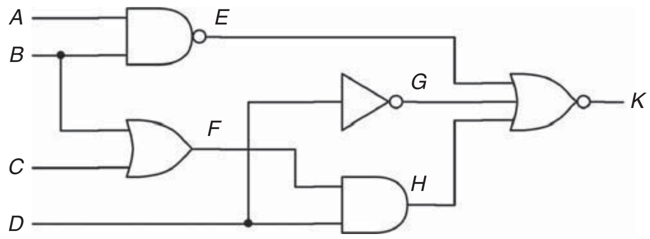


FIGURE 8.45

The example circuit *M*.

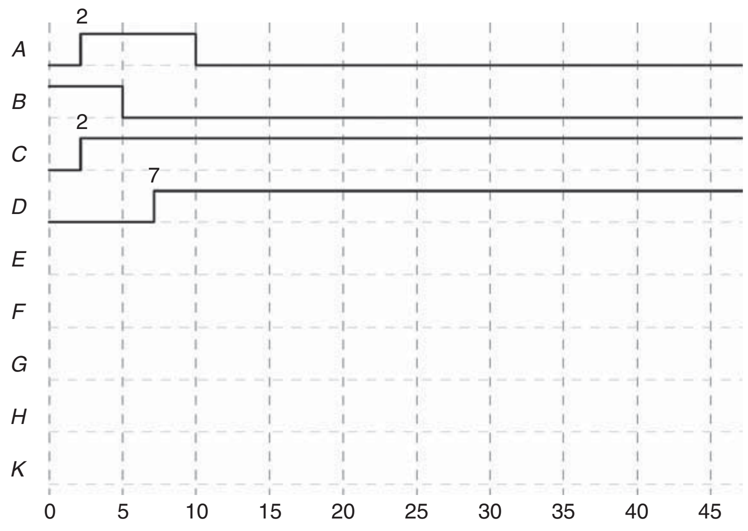


FIGURE 8.46

The timing diagram.

b. *Rise delay*—Two-input gate, 8 ns; three-input gate, 10 ns; inverter, 6 ns.

Fall delay—Two-input gate, 6 ns; three-input gate, 8 ns; inverter, 4 ns.

c. *Minimum delay*—Two-input gate, 8 ns; three-input gate, 10 ns; inverter, 6 ns.

Maximum delay—Two-input gate, 10 ns; three-input gate, 12 ns; inverter, 8 ns.

8.3. **(Compiled-Code Simulation)** One approach to speed up compiled-code simulation is to use the bit-wise logic operations. If two-valued logic is used and the host computer's data word width is 32-bit, one can store in a single word 32 copies of a signal (with respect to different input vectors) and process them at the same time. In this problem, we consider a logic simulator with four logic symbols (0, 1, u, and Z) that are encoded as follows: $v_0 = (00)$, $v_1 = (11)$, $v_u = (01)$, and $v_Z = (10)$. To simulate w input vectors in parallel, two words (X_1 and X_2) are allocated for each signal X to store the first and second bits of the logic symbol codes, respectively.

a. Derive the gate evaluation procedures for AND, OR, and NOT operations.

b. Derive the evaluation procedures for 2-to-1 multiplexer, XOR, and tristate buffer.

8.4. **(Event-Driven Simulation)** Redo Problem 8.2a, using the nominal-delay event-driven simulation technique. Show the event and activity lists of each time stamp.

8.5. **(Interconnection Architectures)** It is known that the full-crossbar chip complexity (Figure 8.20a) grows quadratically with the total pin count. How about the partial-crossbar solution (Figure 8.20b)?

8.6. **(Numerical Integration)** Show that the degree of precision for Simpon's rule is 3. Begin first by considering the integration formula:

$$I_3(f) = \sum_{j=0}^{2} \alpha_j f(x_j)$$

Assuming a uniform time step h, show that the formula is exact for the functions $f(x) = 1, x, x^2$ across the points $x_0 = -h$, $x_1 = 0$, and $x_2 = h$ (this should involve solving a system of three equations for the coefficients α_j). Finally, try these coefficients for higher order polynomials, *i.e.*, $f(x) = x^3, x^4$ to determine the error scaling as a function of h.

8.7. **(Numerical Integration)** With the formula for I_3 derived above, write a simple script to evaluate

$$I_3(f) = \int_{-5}^{5} 5x^4 + 3x^2 - 1$$

Use three different values of the time step h and compare with the expected error scaling determined in Exercise 8.6.

8.8. **(Modified Nodal Analysis)** Formulate the MNA equation representation for the circuit shown in Figure 8.47. The equations should be in terms of the parameters (v_s, R_1, R_2, C_1, C_2) and the unknown voltages and currents (v_1, v_2, i_1, i_2, i_3, i_4).

8.9. **(Modified Nodal Analysis)** Given resistance values: $R_1 = R_2 = 10\ \Omega$ and capacitance values: $C_1 = C_2 = 10^{-10}$ F, use the MNA equations constructed in Exercise 8.8 to plot $v_2(t)$ for a step input from the source v_s.

8.10. **(Wire Capacitance)** Assuming the following dimensions for the two parallel wires shown in Figure 8.48: $h = 1$ μm, $w/h = 2$, $l = 100$ μm, $d = 2$ μm, and $t = 0.75$ μm, compute both the total and coupling capacitances of the wires. Assume that the field terminates as shown

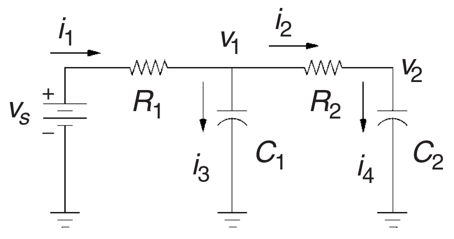


FIGURE 8.47

RLC circuit example.

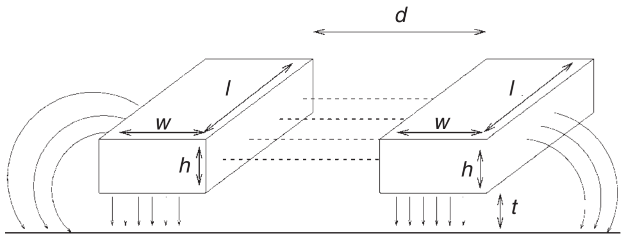


FIGURE 8.48

Parallel wires.

in Figure 8.48. Now, assume that the ratio w/h is doubled. How does this affect the capacitances? Finally, assume instead that the distance between the wires is doubled. How does this affect the capacitances?

8.11. **(Newton's Method)** Write a simple program that implements the Newton's method seen in Equation (8.19). Follow the pseudocode shown in Algorithm 8.1.

Algorithm 8.1 Newton's Method

```
Newton ($f$, $x_0$, $\epsilon$)
1. $n = 0$;
2. $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$;
3. **if** $f(x_{n+1}) \leq \epsilon$ **then**
4.   **return** $x^* = x_{n+1}$;
5. **else** $n = n + 1$; **goto** *step* 2;
```

8.12. **(Newton's Method)** Use the routine from Exercise 8.11 to solve the nonlinear equation (8.18) for the three consecutive time points assuming:

$$\begin{aligned} h &= 10^{-11}\,\text{s}, \\ C &= 1\,\text{F}, \\ R &= 1\ \Omega, \\ I_S &= 1\,\text{A}, \\ n &= 1, \\ V_S &= 2\,\text{V}, \\ V_1 &= 1\,\text{V}, \\ T &= 300\,\text{K}. \end{aligned}$$

Provide the number of iterations and tolerance assumed.

ACKNOWLEDGMENTS

We thank Dr. Jitesh Jain of Purdue University for his invaluable contribution to the development of Sections 8.5 to 8.8. We also thank Dr. Tsung-Hao Chen of Mentor Graphics and Professor Yu-Min Lee of National Chiao Tung University, Jensen Tsai of SpringSoft, Professor Ren-Song Tsay of National Tsing Hua University, Dr. Ming-Yang Wang of Fortelink, and Professor Duncan M. (Hank) Walker of Texas A&M University for reviewing this chapter.

REFERENCES

R8.0 Books

[Atkinson 1989] K. Atkinson, *An Introduction to Numerical Analysis,* John Wiley & Sons, New York, 1989.

[Cheng 1999] C.-K. Cheng, J. Lillis, S. Lin, and N. Chang, *Interconnect Analysis and Synthesis,* John Wiley & Sons, New York, 1999.

[Dunga 2007] M. V. Dunga, W. M. Yang, X. J. Xi, J. He, W. Liu, M. Cao, X. Jin, J. J. Ou, M. Chan, A. M. Niknejad, and C. Hu, *BSIM4.6.1 MOSFET Model—User's Manual,* University of California, Berkeley, CA, 2007.

[IEEE 1076–2002] *IEEE Standard*, *VHDL Language Reference Manual (IEEE Std. 1076-2002),* IEEE Press, New York, 2002.

[IEEE 1463-2001] *IEEE Standard Description Language Based on the Verilog Hardware Description Language (IEEE Std. 1463-2001),* IEEE Press, New York, 2001.

[Keiser 1979] B. E. Keiser, *Principles of Electromagnetic Compatibility,* Artech House, Dedham, MA, 1979.

[Miczo 2003] A. Miczo, *Digital Logic Testing and Simulation,* 2nd ed., John Wiley & Sons, Hoboken, New Jersey, 2003.

[Palnitkar 1996] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis,* Sunsoft, Mountain View, CA, 1996.

[Rabaey 1996] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective,* Prentice-Hall, Upper Saddle River, NJ, 1996.

[Rabaey 2003] J. M. Rabaey, A. Chandrakasan, and B. Nikoli'c, *Digital Integrated Circuits: A Design Perspective,* 2nd ed., Prentice-Hall, Upper Saddle River, NJ, 2003.

[Thomas 2002] D. E. Thomas and P. R. Moorby, *The Verilog Hardware Description Language,* Springer Science, New York, 2002.

[Tsividis 1987] Y. Tsividis, *Operation and Modeling of the MOS Transistor,* McGraw-Hill, New York, 1987.

[Wile 2005] B. Wile, J. C. Goss, and W. Roesner, *Comprehensive Functional Verification,* Morgan Kaufmann, San Francisco, 2005.

R8.1 Introduction

[SystemC 2008] SystemC, http://www.systemc.org, 2008.

[SystemVerilog 2008] SystemVerilog, http://systemverilog.org, 2008.

R8.2 Logic Simulation Models

[Breuer 1972] M. A. Breuer, A note on three valued logic simulation, *IEEE Trans. on Computers*, C-21(4), pp. 399–402, April 1972.

R8.3 Logic Simulation Techniques

[Ulrich 1969] E. G. Ulrich, Exclusive simulation of activity in digital networks, *Communications of the ACM*, 12(2), pp. 102–110, February 1969.

[Wang 1987] L.-T. Wang, N. E. Hoover, E. H. Porter, and J. J. Zasio, SSIM: A software levelized compiled-code simulator, in *Proc. ACM/IEEE Design Automatic Conf.*, pp. 2–8, June 1987.

R8.4 Hardware-Accelerated Logic Simulation

[Babb 1997] J. Babb, R. Tessier, M. Dahl, S. Z. Hanono, D. M. Hoki, and A. Agarwal, Logic emulation with virtual wires, *IEEE Trans. on Computer-Aided Design*, 16(6), pp. 609–626, June 1997.

[Beausoleil 1996] W. F. Beausoleil, T.-K. Ng, and H. R. Palmer, Multiprocessor for Hardware Emulation, U.S. Patent No. 5,551,013. August 27, 1996.

[Dai 1995] W.-J. Dai, L. Galbiati, III, J. Varghese, and D. V. BuiS. P. Sample, Method of Removing Gated Clocks from the Clock Nets of a Netlist for Timing Sensitive Implementation of the Netlist in a Hardware Emulation System, U.S. Patent No. 5,452,239, September 19, 1995.

[Li 1998] J. Li and C.-K. Cheng, Routability improvement using dynamic interconnect architecture, *IEEE Trans. on Very Large Scale Integration Systems*, 6(3), pp. 498–501, September 1998.

[Lin 2002] S. S.-P. Lin, and P.-S. Tseng, Converification System and Method, U.S. Patent No. 6,389,379, May 14, 2002.

[Sample 1999] S. P. Sample, M. Bershteyn, M. R. Butts, and J. R. Bauer, Emulation System with Time-Multiplexed Interconnect, U.S. Patent No. 5,960,191, September 28, 1999.

[Selvidge 1997] C. W. Selvidge and M. L. Dahl, Transition Analysis and Circuit Resynthesis Method and Device for Digital Circuit Modeling, U.S. Patent No. 5,649,176, July 15, 1997.

[Tseng 2001] P.-S. Tseng, S. S.-P. Lin, and Q. K.-H. Shen, Timing-Insensitive Glitch-Free Logic System and Method, U.S. Patent No. 6,321,366, November 20, 2001.

[Varghese 1993] J. Varghese, M. Butts, and J. Batcheller, An efficient logic emulation system, *IEEE Trans. on Very Large Scale Integration Systems*, 1(2), pp. 171–174, June 1993.

[Wang 2006] M. Y. Wang, S. Shei, and V. Chiu, Clock Distribution in a Circuit Emulator, U.S. Patent No. 7,117,143, October 3, 2006.

R8.5 Circuit Simulation Models

[Chen 2003] T.-H. Chen, C. Luk, and C. C.-P. Chen, INDUCTWISE: Inductance-wise interconnect simulator and extractor, *IEEE Trans. on Computer-Aided Design*, 22(7), pp. 884–894, July 2003.

[Ho 1975] C. W. Ho, A. E. Ruehli, and P. A. Brennan, The modified nodal approach to network analysis, *IEEE Trans. on Circuits and Systems*, 22(6), pp. 504–509, June 1975.

R8.7 Simulation of VLSI Interconnects

[Chen 2003] T.-H. Chen, C. Luk, and C. C.-P. Chen, INDUCTWISE: Inductance-wise interconnect simulator and extractor, *IEEE Trans. on Computer-Aided Design*, 22(7), pp. 884–894, July 2003.

[Hoer 1965] C. Hoer and C. Love, Exact inductance equations for rectangular conductors with applications to more complicated geometries, *J. of Research of the National Bureau of Standards*, 69C, pp. 127–137, April-June 1965.

[Jain 2004] J. Jain, C.-K. Koh, and V. Balakrishnan, Fast simulation of VLSI interconnects, in *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 93–98, November 2004.

[Kamon 1994] M. Kamon, M. J. Tsuk, and J. K. White, FASTHENRY: A multipole-accelerated 3-D inductance extraction program, *IEEE Trans. on Microwave Theory and Techniques*, 42, pp. 1750–1758, September 1994.

[Nabors 1991] K. Nabors and J. White, Fastcap: A multipole accelerated 3-D capacitance extraction program, *IEEE Trans. on Computer-Aided Design*, 10(11), pp. 1447–1459, November 1991.

[Wu 1992] R.-B. Wu, C.-N. Kuo, and K. K. Chang, Inductance and resistance computations for three-dimensional multiconductor interconnect structures, *IEEE Trans. on Microwave Theory and Techniques*, 40, pp. 263–270, February 1992.

[Yuan 1982] C. P. Yuan and T. N. Trick, A simple formula for the estimation of the capacitance of two-dimensional interconnects in VLSI circuits, *IEEE Electronic Device Letters*, EDL-3(12), pp. 391–393, December 1982.

[Zhong 2003] G. Zhong and C.-K. Koh, Exact closed form formula for partial mutual inductances of on-chip interconnects, *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 50(10), pp. 1349–1353, October 2003.

R8.8 Simulation of Nonlinear Devices

[Jain 2006] J. Jain, S. Cauley, C.-K. Koh, and V. Balakrishnan, SASIMI: Sparsity-aware simulation of interconnect-dominated circuits with nonlinear devices, in *Proc. IEEE/ACM Asia and South Pacific Design Automation Conf.*, pp. 422–427, January 2006.

[Zhu 2007] Z. Zhu, H. Peng, K. Rouz, M. Borah, C. K. Cheng, and E. S. Kuh, Two-stage Newton-Raphson method for transistor level simulation, *IEEE Trans. on Computer-Aided Design*, 26(5), pp. 881–895, May 2007.

R8.9 Concluding Remarks

[Cadence 2008] Cadence Design Systems, http://www.cadence.com, 2008.

[EVE 2008] EVE, http://eve-team.com, 2008.

[Mentor 2008] Mentor Graphics, http://www.mentor.com, 2008.

[Nagel 1975] L. W. Nagel, *SPICE2*: *A computer program to simulate semiconductor circuits*, Technical Report ERL-M520, University of California, Berkeley, May 1975.

[Stackhouse 2008] B. Stackhouse, A 65nm 2-billion-transistor quad-core itanium processor, in *Proc. IEEE International Solid-State Circuits Conference*, pp. 592–598, February 2008.

[Synopsys 2008] Synopsys, http://www.synopsys.com, 2008.